AVR323: Interfacing GSM modems

8-bit AVR® Microcontrollers

Application Note

Features

- **Interface to GSM modems.**
- **Implementation of AT-Command set.**
- **PDU string compression and decompression.**
- **SMS transmission, how to send and receive.**

1 Introduction

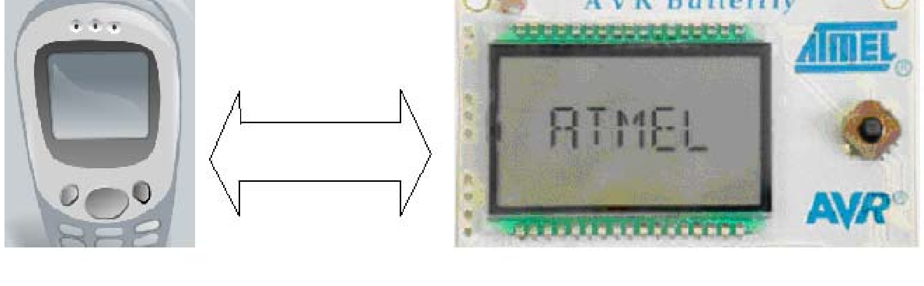
The GSM net used by cell phones provides a low cost, long range, wireless communication channel for applications that need connectivity rather than high data rates. Machinery such as industrial refrigerators and freezers, HVAC, vending machines, vehicle service etc. could benefit from being connected to a GSM system.

Take a given example. A garage offers a very special package to their customers. Based on the mechanics knowledge and the given vehicle, tailored service intervals can be specified. A part of the service agreement is installation of a GSM modem in the vehicle. An onboard service application can then notify the garage when the vehicle approaches its service interval. The garage will schedule an appointment and inform the customer.

The customer will benefit from a reliable and well-serviced vehicle at a minimum cost. The garage on the other hand can provide excellent customer support, vehicle statistics, efficient work scheduling, and minimum stocks.

This application note describes how to use an AVR to control a GSM modem in a cellular phone. The interface between modem and host is a textual protocol called Hayes AT-Commands. These commands enable phone setup, dialing, text messaging etc. This particular application connects an AVR Butterfly and Siemens® M65 cellular phone using a RS232 based data cable. Most cellular phones could be used, except Nokia® phones using F or M-bus.

Figure 1-1. Interconnection phone vs. AVR



Rev. 8016A-AVR-02/06

2 Theory of operation

The protocol used by GSM modems for setup and control is based on the Hayes AT-Command set. The GSM modem specific commands are adapted to the services offered by a GSM modem such as: text messaging, calling a given Phone number, deleting memory locations etc. Since the main objective for this application note is to show how to send and receive text messages, only a subset of the AT-Command set needs to be implemented.

The European Telecommunication Standard Institute (ETSI) GSM 07.05 defines the AT-Command interface for GSM compatible modems. From this document some selected commands are chosen, and presented briefly in this section. This command subset will enable the modem to send and receive SMS messages. For further details, please consult GSM 07.05.

2.1 AT-Command set

The following section describes the AT-Command set. The commands can be tried out by connecting a GSM modem to one of the PC's COM ports. Type in the test-command, adding CR + LF (Carriage return + Line feed = \r\n) before executing. Also see chapter 3.1 for further details.

Table 2-1 gives an overview of the implemented AT-Commands in this application. The use of the commands is described in the later sections.

Table 2-1. AT-Command set overview

Command	Description
AT	Check if serial interface and GSM modem is working.
ATE0	Turn echo off, less traffic on serial line.
AT+CNMI	Display of new incoming SMS.
AT+CPMS	Selection of SMS memory.
AT+CMGF	SMS string format, how they are compressed.
AT+CMGR	Read new message from a given memory location.
AT+CMGS	Send message to a given recipient.
AT+CMGD	Delete message.

Before continuing, the following formats are used in Table 2-2 through 2-9:

- Character string in quotation marks is the actual text sent to modem.
- Optional commands and response parameters are enclosed in brackets.

2.1.1 Status (AT)

The "AT" command is a status request used for testing if a compatible modem is connected and that the serial interface is working properly.

Table 2-2. AT command and possible responses

Command	Response	Comment
"AT"	"OK"	Connected and working
	"ERROR"	Serial line OK, modem error

2.1.2 Echo off (ATE0)

The "ATE0" command is used to config the communication. By default, GSM modems are set to echo any received command back with an acknowledgement. An example of this is shown below.

```
AT\r\n                  //Command sent to modem
AT\r\nOK\r\n            //Response from modem with echo enabled
```

After sending "AT", the modem replies with "AT\r\rOK\r\n". With echo off, "ATE0", the modem would have answered "\r\nOK\r\n" when executing "AT".

The echo off command will reduce traffic on the serial line. The "ATE1" command will enable echo again.

Table 2-3. ATE0 command and possible responses

Command	Response (echo off)	Comment
"ATE0"	"OK"	Echo off
	"ERROR"	Could not turn echo off

2.1.3 New Message Indication (AT+CNMI)

"AT+CNMI" configures how the modem signals arrival of new messages to the connected terminal device and how they are stored in the modem. This feature is useful when it comes to reading new messages. Instead of polling the modem periodically for arrival of new messages, "AT+CNMI" can tell when a new message has arrived. The AVR will catch such indication, and set a flag. This ensures that the modem only takes up CPU resources when necessary.

Table 2-4. AT+CNMI command and possible responses

Command	Response	Comment
"AT+CNMI=[mode][1],[mt][2],[bm][3],[ds][4],[bfr][5]"	"OK"	Mode set
	"ERROR"	Error, could not set such mode.

Notes: 1. [mode] integer type: how messages are buffered.
2. [mt] integer type: indication of new SMS, set to 1.
3. [bm] integer type: Not in use.
4. [ds] integer type: Not in use.
5. [bfr] integer type: Not in use.

What values “[mode]”, “[mf]”, “[bm]”, “[ds]” and “[bfr]” could take will be different from modem to modem. This should be tested off line with modem connected to the PC. An example is given below:

```
AT+CNMI=?\r\n                             //Possible value request

+CNMI: (0,1),(0,1),(0,2),(0,2),(1)    //Possible parameter values

OK                                        //Command executed OK
```

2.1.4 Preferred Message Storage (AT+CPMS)

The “AT+CPMS” command sets the target memory location for storing sent, read, deleted and received SMS messages. Most modems have multiple storage types:

- “SM”: SIM card memory.
- “ME”: Mobile Equipment storage. Dedicated storage within the modem for text messages only.
- “MT”: Collection of all storage connected to the modem: SM, ME or others. The phone will chose one appropriate if this option is enabled.

Table 2-5. AT+CPMS command and possible responses

Command	Response	Comment
“AT+CPMS=[M1][1],[M2][2],[M3][3]”	“+CPMS:[used1],[total1],[used2],[total2],[used3],[total3] \r\rOK\r\n”	Memory configured OK
	“+CMS ERROR”	Error

Notes: 1. [M1] string type: Memory from which messages are read and deleted.
2. [M2] string type: Memory to which messages are written and sent.
3. [M3] string type: Memory in which received messages are stored, if forwarding to pc is not set.

[used] integer type: is number of messages currently in x.

[total] integer type: is total number of message locations in x.

2.1.5 Message format(AT+CMGF)

The “AT+CMGF” command is used to set input and output format of SMS messages. Two modes are available:

- PDU mode: reading and sending SMS is done in a special encoded format.
- Text mode: reading and sending SMS is done in plain text.

PDU mode is described later in section 2.2. This compressed format saves message payload and is default on most modems. PDU mode is implemented in the source code for this application note, it is possible to use text mode to reduce code footprint if the connected modem supports this.

In text mode header fields as sender address, message length, validation period etc. can be read out in plain text together with the sent message. Please consult GSM

07.05 for more about reading messages in text mode. This is not the main target for this application.

Table 2-6. AT+CMGF command and possible responses

Command	Response	Comment
“AT+CMGF=[mode][1]”	”OK”	Mode selected
	“ERROR”	Error

Notes: 1. [mode] integer type: 0 is PDU mode, 1 is text mode.

2.1.6 Read Message(AT+CMGR)

The “AT+CMGR” command is used to read a message from a given memory location. Execution of “AT+CMGR” returns a message at [index] from selected memory [M1] (See section 2.1.4 for memory setup). The status of the message and the entire compressed message (PDU) is returned. To get any useful information out of the compressed message it should be decompressed. The PDU format and the compression and decompression is described in section 2.2.2.

Table 2-7. AT+CMGR command and possible responses

Command	Response	Comment
“AT+CMGR=[index][1]”	“+CMGR:[stat][2],[alpha][3], [length][4] \r\n [pdu][5] “	Message read OK
	“+CMS ERROR”	Error, No such index

Notes:
1. [index] integer type: Read message from location [index].
2. [stat]: integer type: Status of message in memory: READ, UNREAD, SENT and UNSENT.
3. [alpha] integer type: Manufacturer specific field. Not used.
4. [length] integer type: Length of compressed message.
5. [pdu] string type: Compressed message.

2.1.7 Send Message(AT+CMGS)

This command enables the user to send SMS messages. Section 2.2.3 describes how to build such messages. How to include user defined text and recipient telephone number. After the user defined fields are set, the message can be compressed and sent using the “AT+CMGS” command. An example usage of “AT+CMGS” is given in section 2.2.3.

Table 2-8. AT+CMGS command and possible responses

Command	Response	Comment
"AT+CMGS=[length][1] CR[2] [pdu] [3] ctrl-Z[4]"	"OK"	Message sent
	"+CMS ERROR"	Command error

Notes: 1. [length] integer type: Length of message.
2. CR = Carriage return
3. [pdu] string type: Compressed message
4. Ctrl-Z: Command terminator. ASCII character 26 (dec).

2.1.8 Delete Message(AT+CMGD)

This command is used to delete a received stored message from [M1] (See Table 2-5).

Table 2-9. AT+CMGD command and possible responses

Command	Response	Comment
"AT+CMGD=[index][1]"	"OK"	Message deleted
	"ERROR"	Command error

Notes: 1. [index] integer type: Index of message to delete.

This concludes the presentation of the implemented AT-Command set. More commands are discussed in ETSI standard GSM 07.05, and proposed as a reference when working with applications interfacing GSM compatible modems together with manufacturer's datasheet.

2.1.9 Error codes

Many of the commands in the implemented subset can terminate with an error message related to the modem or network. These could be errors such as:

- Memory failure.
- Invalid recipient number.
- Network timeout.
- SIM busy or wrong.
- Operation not allowed.
- No network service.

These error messages can be useful, and could be implemented as a part of the application. It is possible to extend the handling of the error codes, but this is beyond the scope of this application note. We will just catch the ERROR message, and repeat the command.

If more advanced error handling is desired one should refer to the modem datasheet.

2.2 PDU format explained

There are two ways of sending and receiving SMS messages: by text mode and by PDU (Protocol Description Unit) mode. By default most phones and modems are setup to send SMS messages using a special compression format (PDU-mode).

Some modems supports text-mode, in which any information and the message itself can be read as plain text. Note however that not all phones and modems support text-mode.

2.2.1 Special data types and compression/decompression

PDU-mode uses three special data types:

- Octet: Group of 8 bits in hexa-decimal encoding (0x00→0xFF). Example: **E8**.
- Semi-octet: Group of 8 bits in decimal encoding (0→153). Example: **11**.
- Septet: Group of 7 bits in integer encoding (0->127). Example: **126**.

The default GSM alphabet uses 7 bits to represent characters. The message “hello” consists of five characters called septets, when represented with seven bits each. The septet string has to be encoded into an octet stream for SMS transfer (See Table 2-10).

Table 2-10. Compressing septet string into octet stream

Value	**h**	**e**	**l**	**l**	**o**
Decimal	104	101	108	108	111
Hex	0x68	0x 65	0x 6C	0x 6C	0x 6F
Septet	1101000	110010**1**	11011**00**	1101**100**	110**1111**
8-bit	**1**1101000	**00**110010	**10**011011	**1111**1101	**00000**110
Octet	E8	32	9B	FD	06

Notes: The first septet (h) is turned into an octet by adding the rightmost bit of the second septet (Bold). Inserted at the left this gives: 1 + 1101000 = 11101000 ("E8").

The second char (septet) then receives two bits (bold) from the third septet, so the second character (e) become an octet: 00 + 110010 = 00110010 ("32").

The five first bits of the last char (o) is padded with zeros (bold).

Messages encoded this way can then be added as payload to “AT+CMGS” command described in section 2.1.7.

When receiving a new message, “AT+CMGR” can be used to read from the memory location where it resides. An octet stream will be returned from the modem.

To extract any useful information from this stream a decompression method is needed. Table 2-11 shows an example decoding the octet representation of “hello” back into septets.

Table 2-11. Decompressing octet stream into septets

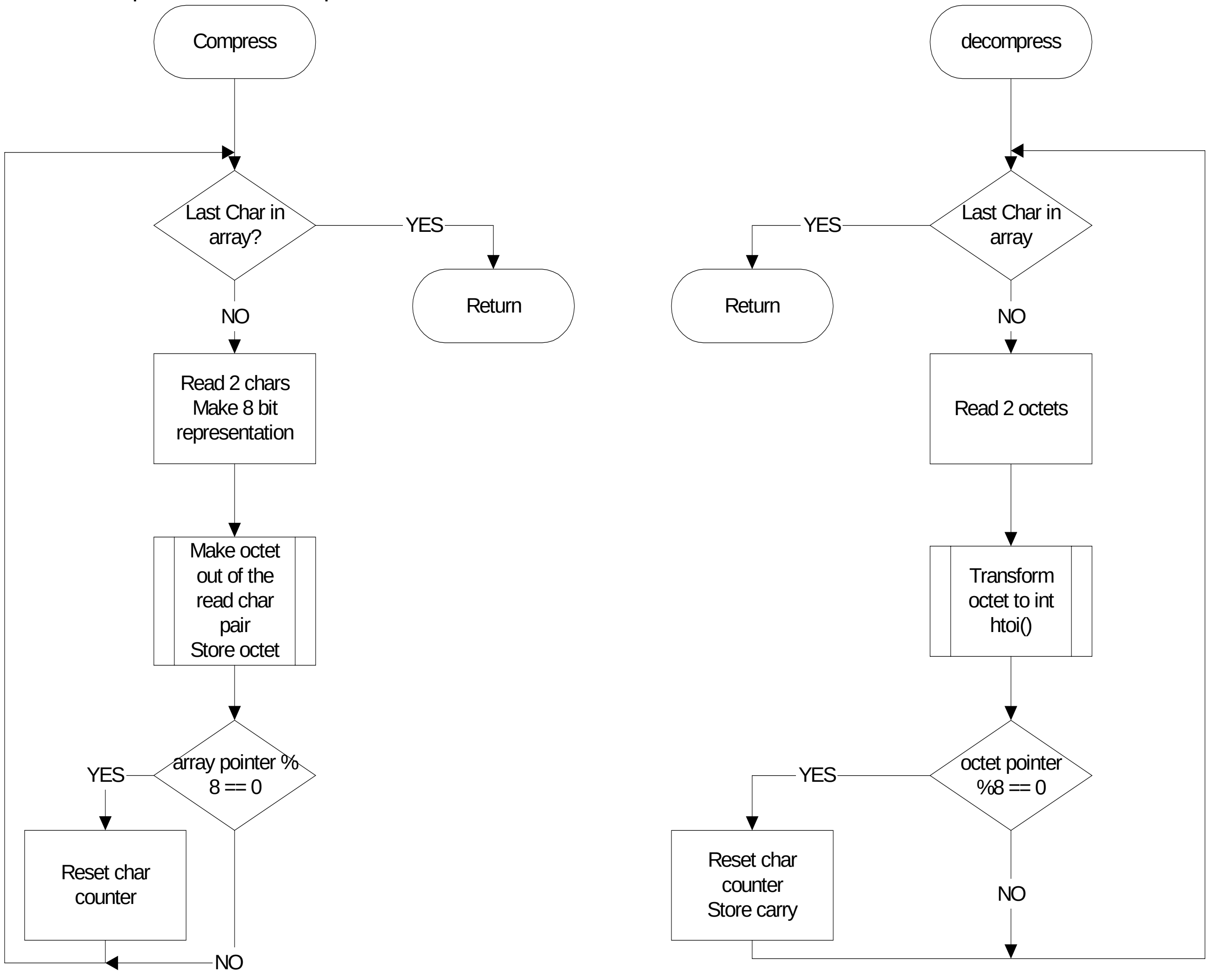
Octet	**E8**	**32**	**9B**	**FD**	**06**
8-bit	**1**1101000	**00**110010	**10**011011	**1111**1101	**00000**110
Septet	1101000	110010**1**	11011**00**	1101**100**	110**1111**
Decimal	104	101	108	108	111
Value	h	e	l	l	o

Notes: To become a septet, the first octet looses its leading 1(Bold) to the second octet. Here it is added at the back, meanwhile the two leading zeros are discarded.

The last octet (06) looses its leading zero padding and copies (1111) received from the 4'th octet at its end.

Figure 2-1 shows flowcharts for both routines as implemented in the source code. Algorithms are based on heuristic methods, since no such is specified in GSM 03.38 or GSM 03.40.

Figure 2-1. Compress and decompress flowchart



2.2.2 Receiving a message in PDU mode

A SMS string consists of mainly three parts: length of header, header and the PDU string. When reading a message from the modem using "AT+CMGR", a SMS string should be received:

```
AT+CMGR=42        //Read out SMS from message storage 42

+CMGR: 0,,42      //Received from phone
                  //SMS string:
07914467429499400040ED0C5BAFC2D0ED3CB0000504062319491401 9E8329BFD06B5
40A06B10EA2A56A54F61905A740D9F4D

OK   //Acknowledge from phone. AT+CMGR returned OK.
```

To most people the above code snippet does not contain any readable information at all. Table 2-12 shows how to extract details from the returned SMS string.

For a thorough presentation of all fields, code schemes, alphabets etc., please consult GSM 03.40.

Table 2-12. Details in SMS string.

	Description	Description
Header	**07**	Number of octets in header, 0x07 = 7
	91	Numbering plan. 91 is international
	446742949940	Service center number as semi-octet and reversed. Real number is +447624499904
PDU-String	**04**	First octet of SMS-Deliver. Message sent from service center to GSM modem.
	0E	Length of address 0x0E = 14
	D0	Address type
	C5BAFC2D0ED3CB	Sender: "Eurobate", octet string
	00	Protocol identifier
	00	Coding scheme
	50406231949140	Timestamp, semi-octets: 26.04.05 13:49:19 GMT+1,00
	19	Length of user data. 0x19 = 25 septets
	E8329BFD06B540A06B10EA2A56A54F61905A740D9F4D	User defined text: "**hello-WAP.EUROBATE.COM**".

2.2.3 Building and sending a message in PDU mode

The SMS message, as specified by the ETSI organization, can be up to 160 septets long. The maximal user payload is then restricted to 140 octets, together with additional fields in the PDU protocol. These additional fields are crucial since they contain information about receivers-address, address length, validity-period, type of address, data coding scheme, protocol identifier etc.

A message sent from modem to a service center is called an SMS-SUBMIT message. Table 2-12 shows how to build such messages. To avoid any problems with manufacturer specific meta-data, the modem is set to do this using the “00” option. Rest of the protocol stack is defined according to GSM 03.40.

Table 2-13. SMS-SUBMIT message fields.

	Octet	Description
Header	**00**	Number of octets in meta data, 0 means that modem should use stored meta.
PDU-String	**11**	First octet in SMS-SUBMIT
	00	Message reference, 00 means that the modem sets the reference number.
	0A	Address length: 0x0A = 10
	91	Number type, international type
	7421436587	Address of receiver: +4712345678
	00	Protocol identifier
	00	Data coding scheme
	AA	Expire time: 4 days
	05	Septet count: 0x05 = 5 septets.
	E8329BFD06	User defined text: hello.

In advance of sending our string, we need to calculate its length. Counting the number of octets, excluding the leading meta-information, gives a total length of 17 octets for the SMS string in Table 2-13.

Using a terminal application, the following could be sent to the modem.

```
AT+CMGS = 18      //Send a message containing 18 octets, excluding
                  //the two initial zeros
```

The modem will now delay for a while and get ready to receive the 18 octets long SMS string. A “\r\n> ” will be displayed on the screen when the modem is ready to append your payload. See code below.

```
>                 //Promt given from phone when ready to send
0011000A9174214365870000AA05E8329BFD06<ctrl-z> //SMS string to send
```

The modem should now return “OK”, meaning that the message is sent. Using a Cellular Phone this could be verified as a new entry in the Sent Items folder.

Most error messages for “AT+CMGS” originate from using the wrong message length, so this should be checked twice. Remember to omit counting any leading zeros in the header!

3 Interfacing the GSM modem from a PC

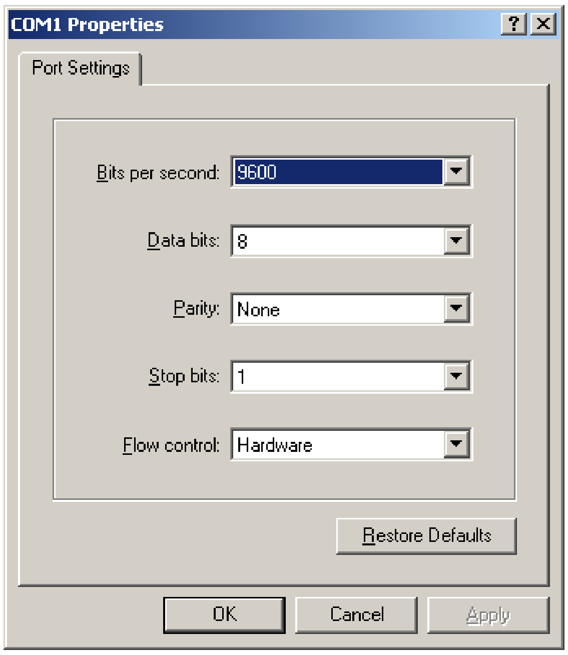
All commands given in chapter 2.1.1 - 2.1.8 can be tested having a GSM compatible modem connected to a PC using a suitable data cable.

3.1 Hardware setup and communication settings

To test the available modem and how it responds to AT-Commands, connect it to a PCs COM port. This application note assumes that the phone will be connected using a RS232 data cable, though IrDA® could be used if available.

With the phone connected, open a terminal application. Communication settings should be found in the modem datasheet. If no such information can be obtained, try the ones in Figure 3-1.

Figure 3-1. Communication settings


COM1 Properties

Port Settings

Bits per second: 9600

Data bits: 8

Parity: None

Stop bits: 1

Flow control: Hardware

Now the connected system should enable sending AT-Commands from the terminal window. Test with "AT" to verify this.

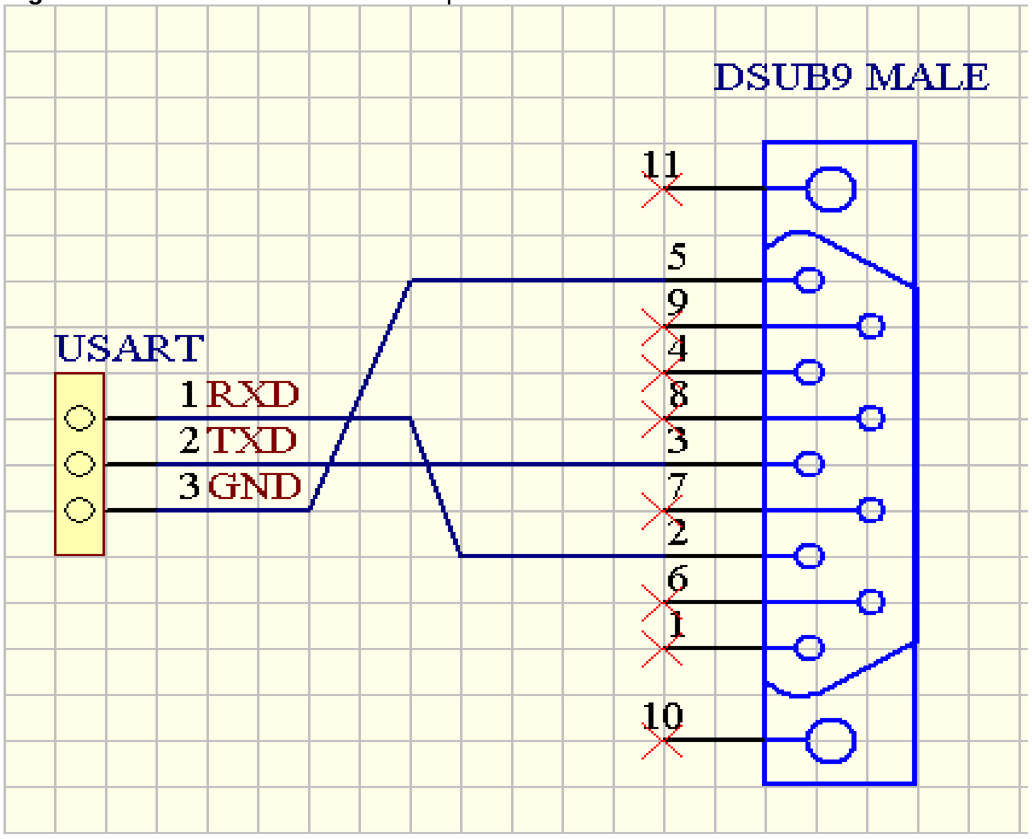
Connecting the same RS232 data cable to the AVR Butterfly, a suitable adapter has to be made. Outputs from the level-shifter on the AVR Butterfly are routed to a 3x1 header, and not directly compatible with the RS232 cable.

An adapter is easily made out of a male DSUB9 connector and two 2-wire cables (Supplied with the STK500). Pin-out and wiring for such an adapter is shown in Table 3-1and Figure 3-2.

Table 3-1. Pin chart for USART connection.

D-sub 9 male pin number	AVR Butterfly	Comment
2	RX	Twist with TX
3	TX	Twist with RX
5 (GND)	GND	GND

Figure 3-2. Schematics for serial adapter



3.2 Example setup using PC

Table 3-2. Phone setup explained

Command sent	Received from modem	Comment
”AT”	”AT” ”OK”	Modem present, and compatible with AT-Commands
	“ERROR”	Phone not connected
”ATE0”	”ATE0” ”OK”	Echo off
	“ERROR”	Phone not connected or erroneous command
“AT+CPMS=”ME”,”ME”,”ME””	“+CPMS:” “OK”	Memory selection was successful
	“ERROR”	One or more memory locations not available

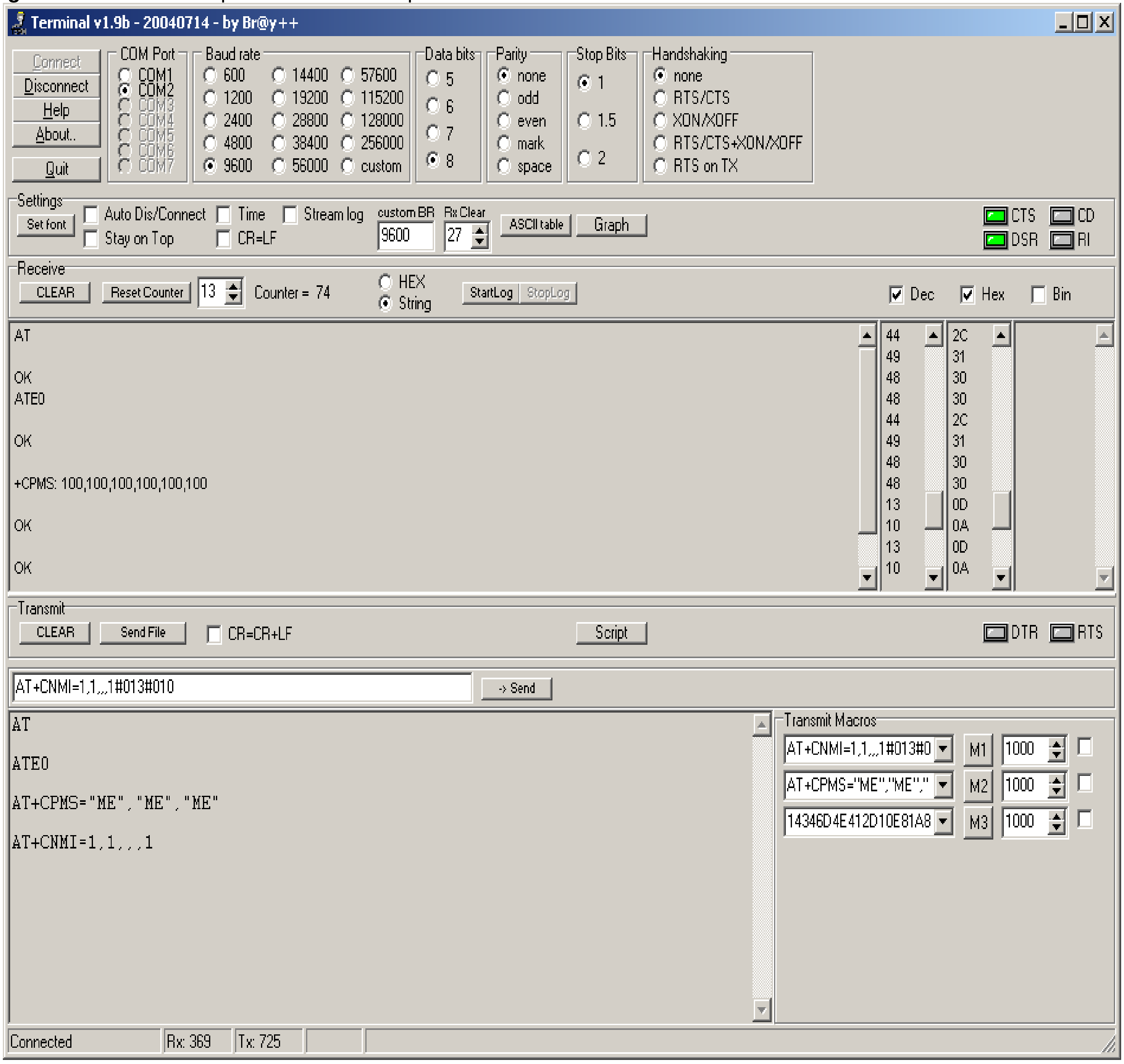
”AT+CNMI=1,1,0,0,1”	”OK”	New message indication enabled
	“ERROR”	Could not enable this mode

Table 3-2 Shows how to setup the connected modem using a terminal application. Type the contents in the “Command Sent” column, appending CR+LF at the end.

If you receive any error messages, check your command for exact spelling and parameters. The datasheet for the connected modem may contain more information about the error messages, and be a great tool when debugging.

In Figure 3-3 details from modem setup using Bray Terminal application can be viewed.

Figure 3-3. Screen dump from modem setup



COM Port: COM1 [✓ COM2] COM3 COM4 COM5 COM6 COM7

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

Settings: Auto Dis/Connect, Stay on Top, Time, CR=LF, Stream log; custom BR 9600; Rx Clear 27

Receive: 13 Counter = 74; HEX [✓ String]; [✓ Dec] [✓ Hex] Bin

```
AT

OK
ATE0

OK

+CPMS: 100,100,100,100,100,100

OK

OK
```

```
44   2C
49   31
48   30
48   30
44   2C
49   31
48   30
48   30
13   0D
10   0A
13   0D
10   0A
```

Transmit: CR=CR+LF

```
AT+CNMI=1,1,,,1#013#010
```

```
AT

ATE0

AT+CPMS="ME","ME","ME"

AT+CNMI=1,1,,,1
```

Transmit Macros

AT+CNMI=1,1,,,1#013#0	M1	1000
AT+CPMS="ME","ME","	M2	1000
14346D4E412D10E81A8	M3	1000

Connected | Rx: 369 | Tx: 725

4 Implementation

This application note is accompanied with an implementation source code. This source code is made as an example of how to:

- Setup modem for new message indication.
- Send SMS messages containing user-defined text.
- Mechanism for identification of new message received.
- Read SMS message from a given memory location.

Any AVR with sufficient memory capabilities (See section 4.3 for code foot print) and a UART is able to run this application. The source code is contained in five files and their headers. Together this forms a library that enables most Atmel AVRs to benefit the services from the GSM net.

Source code files:

- **AVRGSM_api.c:** File containing api functions to init, delete, send and read from modem.
- **AVRGSM_com.c:** Low level USART driver.
- **AVR_SMS_tools.c:** Code for extracting important parameters in modem reply.
- **AVR_SMS_zip.c:** Compress and decompress functions for PDU-string.

Further information can be obtained from the Doxygen documentation found in the source code directory.

4.1 System initialization

To initialize the connected modem, the application running on the AVR Butterfly must go through the steps outlined in Table 3-2. But before further details on modem setup, some word about timing.

There is no way for the application to know if the modem for some reason is disconnected. No handshake is implemented so issuing a command could result in lost acknowledgement. The solution is to start a counter when a new AT-Command is sent. In “AVRGSM_com.h” a user configurable timeout level, “RX_WAIT”, is defined. If no “OK” string is received within this timeout period, an error state occurs and prevents endless loops resulting from disconnected modem. Another solution is to implement handshake or have a dedicated phone present pin.

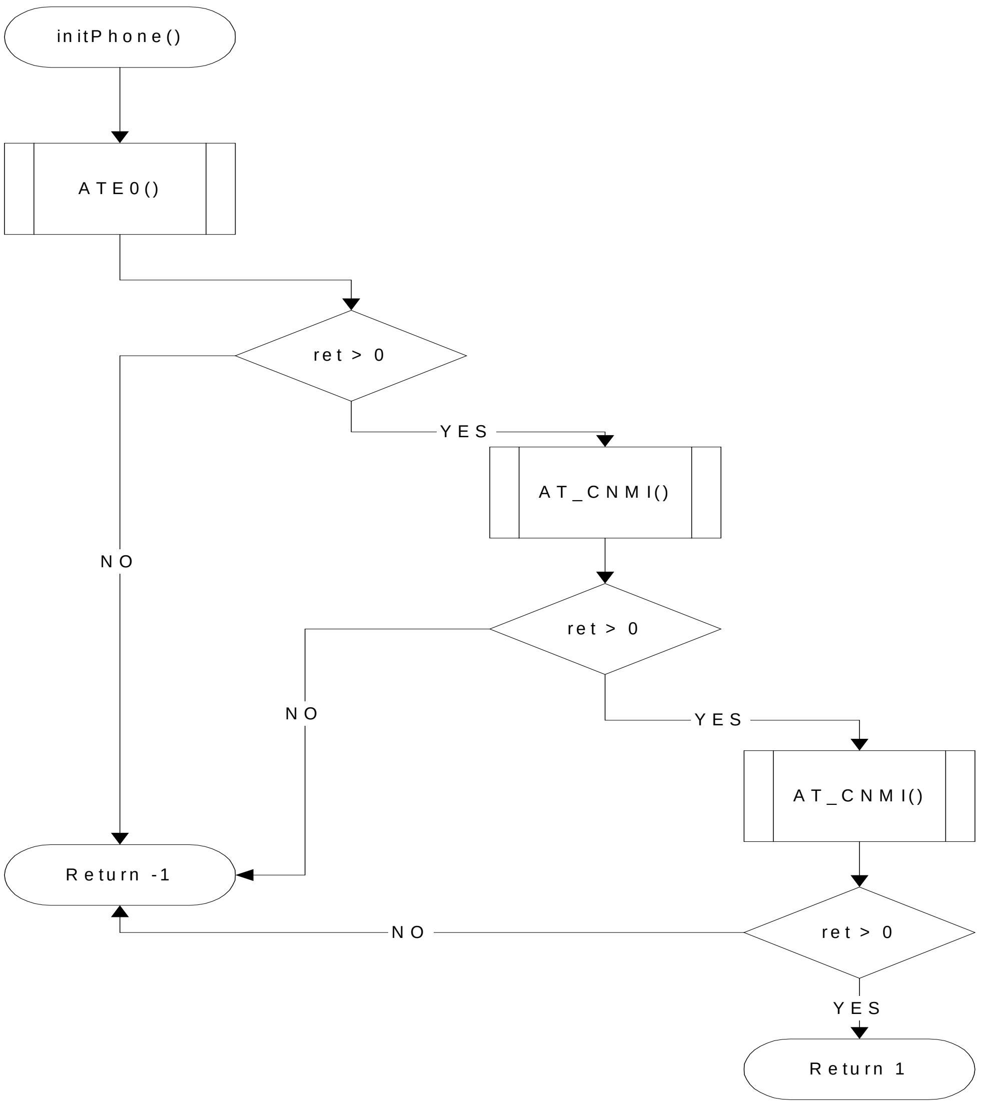
The following pseudo code shows how a message is sent. See also “AVRGSM_api.c” for further documentation.

```
Send AT-Command using printf(<YOUR COMMAND>);
Start timer;
     WAIT for timer expiration or “OK” from modem
     If(Timer Expired || “Error” received)
            Return –1;
     If(“OK”)
            Return 1;
End
```

Sending AT-Commands in this manner the application code can safely determine the modem state.

Continuing with the modem setup, one need to send “ATE0” to turn echo off. If the modem acknowledges with “OK” everything is fine, and next command from Table 3-2 can be executed. If any error occurs the setup routine returns an error code. See Figure 4-1 for complete flowchart for “API_modem_init()” method found in “AVRGSM_api.c”.

Figure 4-1. API_modem_init() flow chart



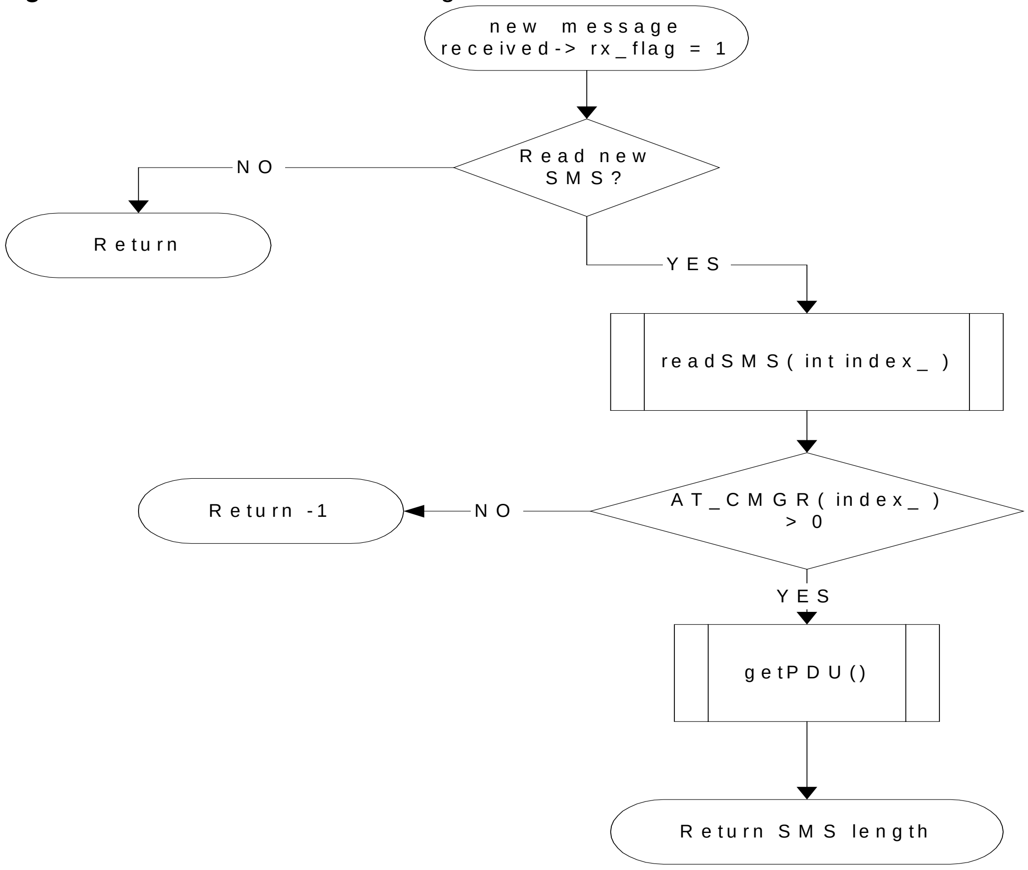
4.2 New message indication

The modem is now setup in such manner that if a new message is sent to the modem, it will notify the Butterfly application and tell that a new message has arrived at memory location [index].

“AT+CNMI=1,1,0,0,1” tells the modem to store any new messages at the preferred storage (see section 2.1.4). The modem is also set to send “+CMTI: <mem>,<index>” whenever a new message arrives. “<mem>” indicates where the new message is stored, and <index> is a pointer to a specific memory index.

Figure 4-2 show what happens after a new message has arrived. “API_readmsg()” will use “AT+CMGR” to read from the given memory index. The contents of the receive buffer is then feed to the “ZIP_decompress()” which extracts the PDU-string just fetched, returning it as readable text.

Figure 4-2. Flowchart for new messages arrived



4.3 Code footprint and compiler settings

In Table 4-1 the code footprint for the five source files is presented with optimization enabled and disabled.

Table 4-1. Code footprint with and without optimization

File	Optimization enabled	Optimization disabled
AVRGSM_api.c	479 bytes	687 bytes
AVRGSM_com.c	602 bytes	785 bytes
AVRGSM_tools.c	154 bytes	274 bytes
AVRGSM_zip.c	557 bytes	669 bytes
Total code size:	**1792 byes**	**2415 bytes**

The following settings need to be defined in the dialog window found under "Project" – "Options". Note that all settings are already defined in the example application.

Table 4-2. Required compiler settings

Category	Tab	Set to	Example
General	Target	Set "Processor configuration" to match target AVR.	-cpu=m169, Atmega169
		Set "Memory model" to Small.	
	Library Configuration	Check "Enable bit definitions in I/O-include files".	
XLINK	Output	Define output file format such that the file can be opened in AVRStudio. Set to ubrof 8.	Ubrof 8 (forced)

5 Known Issues

This application note has been written for a Siemens M65 cellular phone. The code will work for this device as is. Any other GSM modem supporting the AT-Commands interface can also be used, but be aware of the following issues:

- Try the "AT+CNMI" command on your GSM modem. Some modems do not support empty parameter fields: "AT+CNMI=1,1,,,1" (",,," represents three empty fields where the modem should use default settings).
- Check the modem datasheet for correct command terminator. Some modems can manage with just "\r" as terminator, other need "\r\n".

To find and verify the settings for the modem at hand, connect it to a PC as described in chapter 3. Go through all commands that your application need and check how they work and how they respond. Use this as a starting point to modify the application note source code.

6 References

Following documents from www.etsi.org:

- GSM 03.38
- GSM 03.40
- GSM 07.05

Utilized software:

- **pduspy.exe**: Used for verification of SMS-strings.
  http://www.nobbi.com/download.htm
- **Terminal by Bray**: Very stable terminal application.
  http://bray.velenje.cx/avr/terminal