Get more out of the GSM Short Message Service

by Leigh Morris Oct 2012

Abstract

This article shows how to use GSM message PDU mode and improve upon the text mode length limit for an SMS (Short Message Service) text message when using the e-blocks or MIAC System GSM modem.

Requirements

Software:

- Flowcode is preferable, but not essential

Hardware:

- E-blocks upstream board or MIAC Unit
- EB066 GSM Board, or MIAC GSM Expansion module

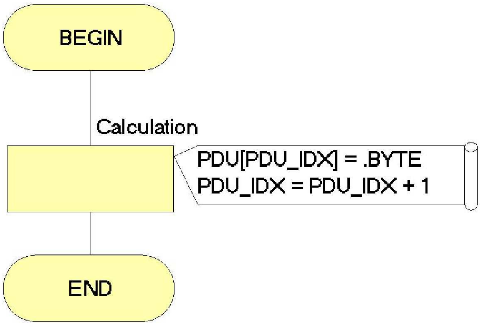
Introduction

The GSM SMS (Short Message Service) text messaging communication payload is limited to 140 bytes, or octets. Short messages can be encoded using a variety of coding schemes and alphabets, these being, the 8-bit data alphabet, the GSM 7-bit alphabet and the 16-bit UCS-2 alphabet (required for such as Arabic and Cyrillic alphabet languages).
Hence this gives the maximum individual short message sizes of 160 7-bit characters, 140 8-bit characters, or 70 16-bit characters. Clearly the GSM 7-bit alphabet allows the most characters to be transferred and support for it is mandatory for GSM handsets. The GSM7 character set is similar to ASCII, but there are some exceptions. For characters not available in the initial code page there is a second code page scheme using an escape character mechanism.

So, how do I send 160 characters?

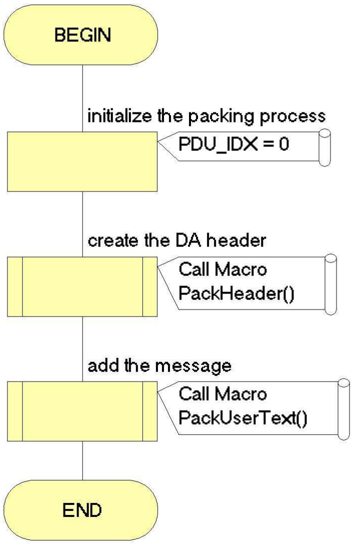
Some modems allow a message to be encoded and sent in GSM7 format, but we can ensure this is the case by using the modem in PDU mode, rather than Text mode. First of all we will assume that all the text characters that are to be sent are available in the GSM7 alphabet.
Next we need to pack the (up to) 160 characters of our message into a maximum size 140 octet buffer. But as we will be working in PDU mode, there is some header information we need to insert first.
So we are going to create an image of the PDU we are sending in a byte array. In Flowcode this is best done by creating a string (of length 200 bytes) PDU[200] and we will need a pointer into the array, so create a byte PDU_IDX for that purpose.

To help insert bytes into the PDU buffer, we create a macro **PackByte(BYTE)**, which simply inserts the BYTE at the current writing index and then increments the index ready for the next write to buffer.



```
PDU[PDU_IDX] = .BYTE
PDU_IDX = PDU_IDX + 1
```

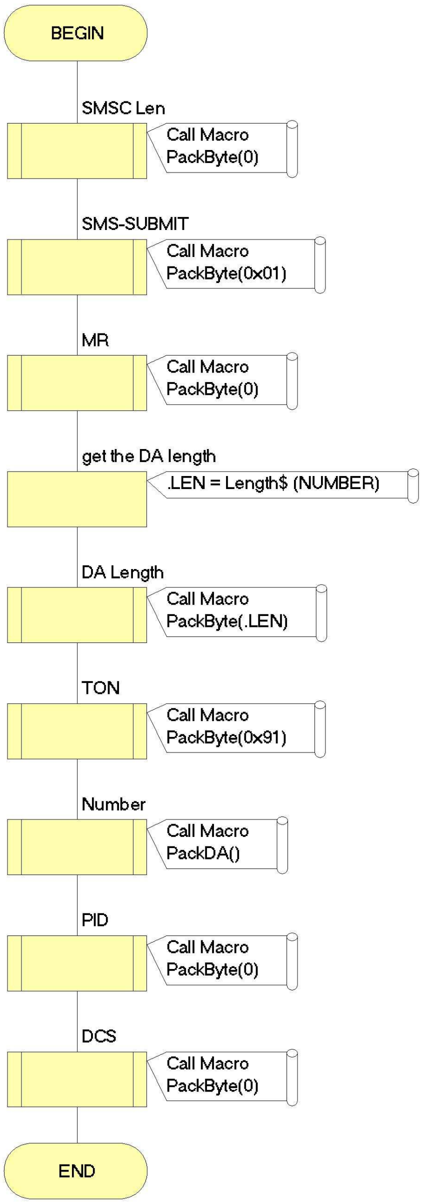
Next we create a macro **PackPDU**, which builds the PDU for the message, as follows:



This separates the PDU process into two main parts, the header and the message body (user text).

Creating the message Header

To create and pack the required message header, we create a global string **NUMBER[20]** to hold the destination number, and a **PackHeader** macro as follows:



1) The first byte indicates the length of the SMSC address, which is zero to force the modem to use the default for the network.

2) Indicate that this is a Submit PDU.

3) Message Reference is set to zero, the modem will use its own reference.

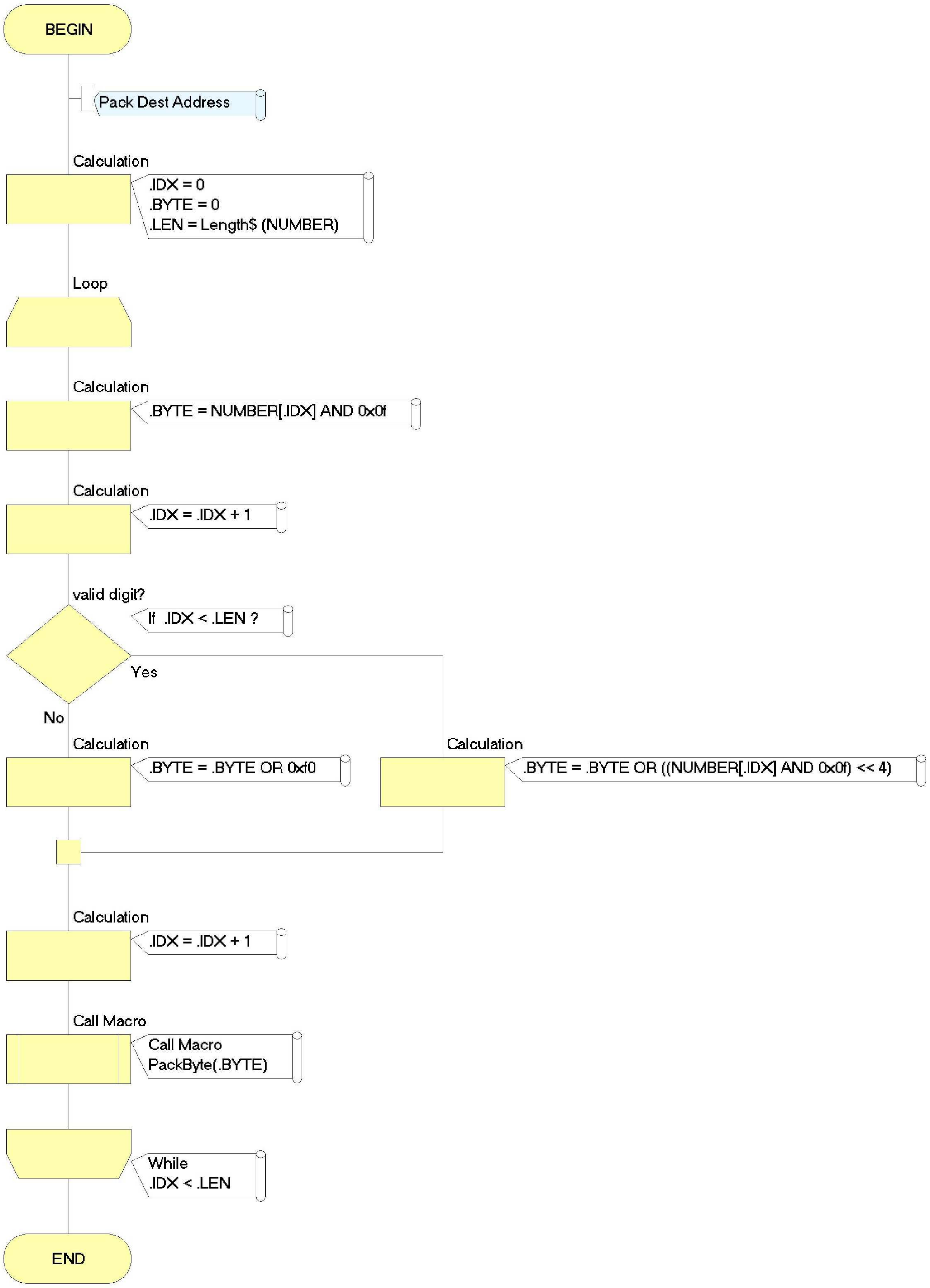
4) Calculate and insert the length of the destination number.

5) Type Of Number set to International Format

6) Protocol Identifier set to zero

7) Data Coding Scheme zero for GSM7

The destination address number needs to be packed into the PDU as two digits per byte, so we create a separate macro to do this, namely **PackDA,** as follows:



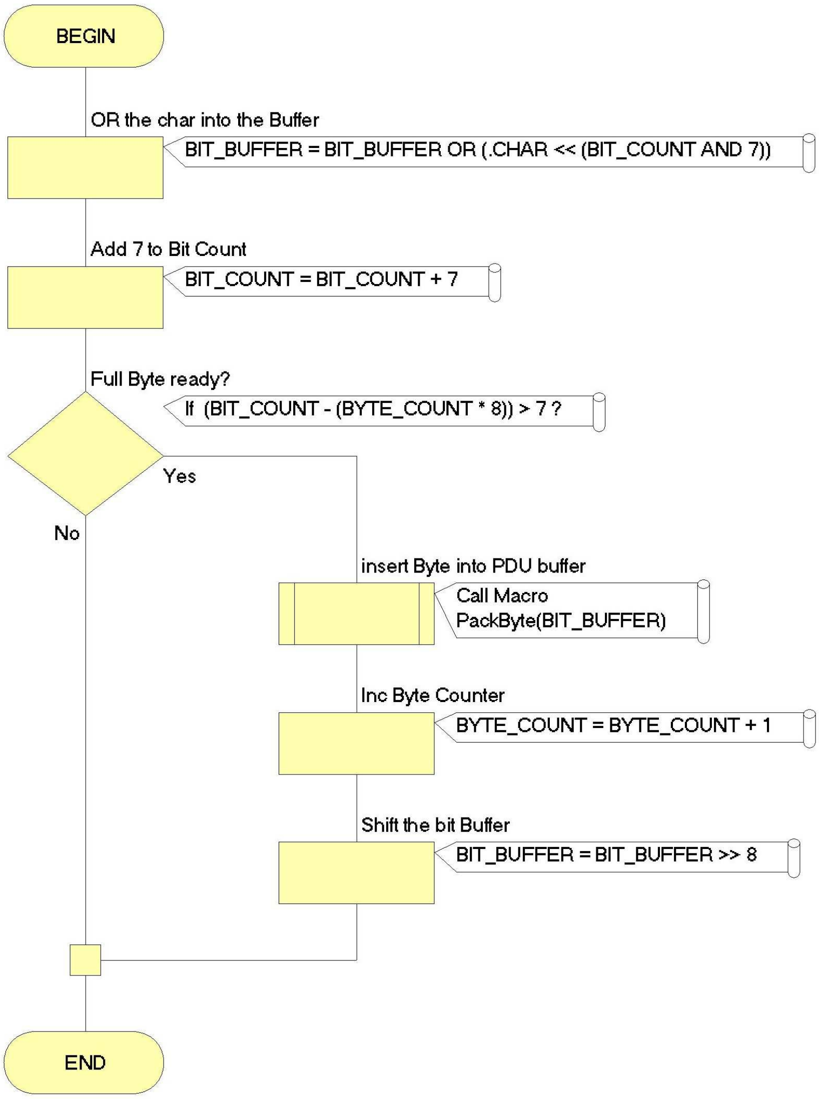
You will notice that if the destination address is an odd number of digits then we are required to insert an extra 4 bits, set to 0x0f0.

Packing the message body (User Text)

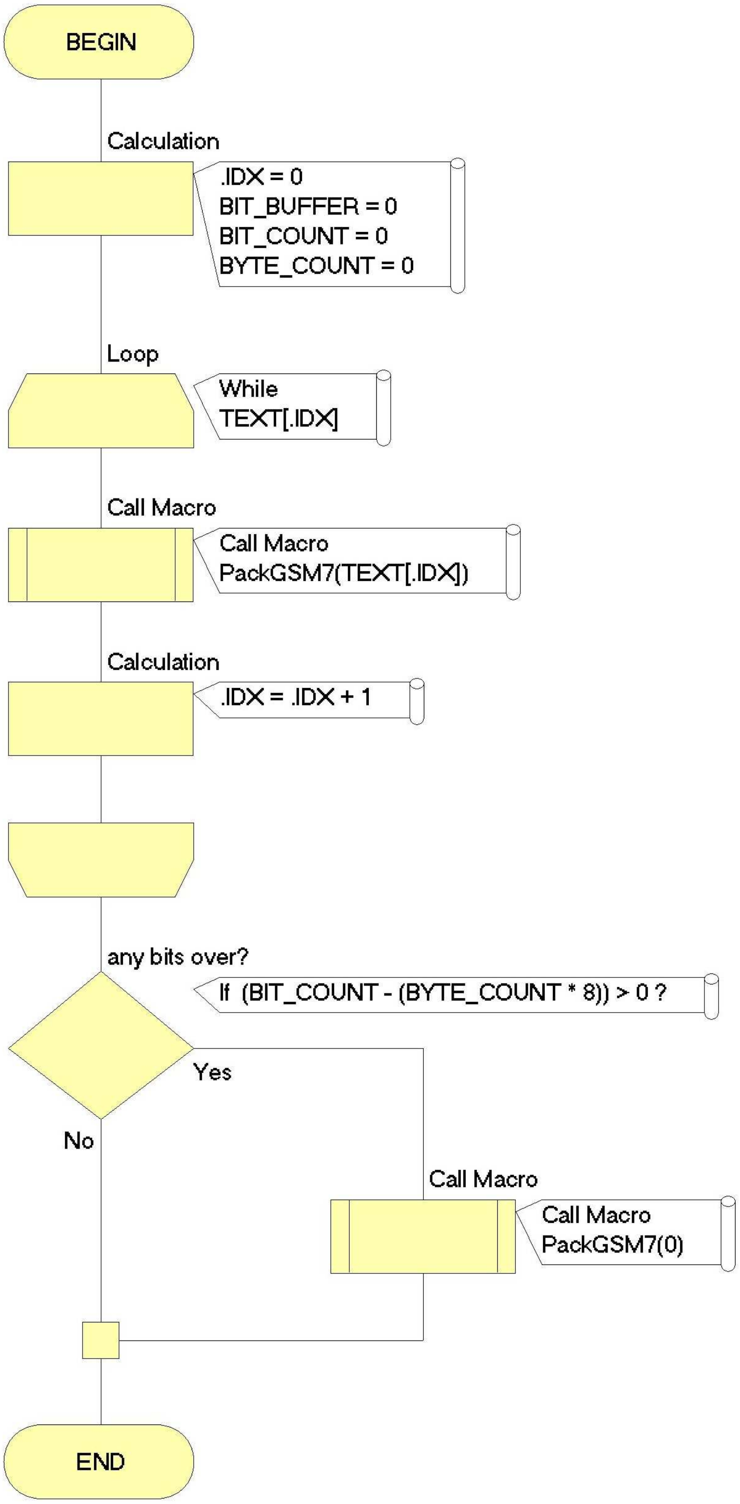
For GSM7 text encoding, the 7 bit wide text characters, or septets, are packed into the octet buffer as viewed as a continuous bit buffer. Hence the free last bit of the first octet in the buffer is used by the first bit of the second septet character and so on. So the septet characters will be split across octet boundaries, until we get to the end of the 8th character, at which point septets and octets will be synchronized, as 8 characters fit into exactly 7 octets.

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Third Octet								Second Octet								First Octet							
			Third Septet							Second Septet							First Septet						

To encode the text we will use a global **UINT BIT_BUFFER** as a shift register and a bit counter **UINT BIT_COUNT**. The macro **PackGSM7(CHAR)** to pack each character into the PDU buffer is as follows:


BEGIN
OR the char into the Buffer
BIT_BUFFER = BIT_BUFFER OR (.CHAR << (BIT_COUNT AND 7))
Add 7 to Bit Count
BIT_COUNT = BIT_COUNT + 7
Full Byte ready?
If (BIT_COUNT - (BYTE_COUNT * 8)) > 7 ?
Yes
No
insert Byte into PDU buffer
Call Macro
PackByte(BIT_BUFFER)
Inc Byte Counter
BYTE_COUNT = BYTE_COUNT + 1
Shift the bit Buffer
BIT_BUFFER = BIT_BUFFER >> 8
END

We can now pack the message text into the PDU buffer by creating a macro **PackText** that uses the **PackGSM7** macro:

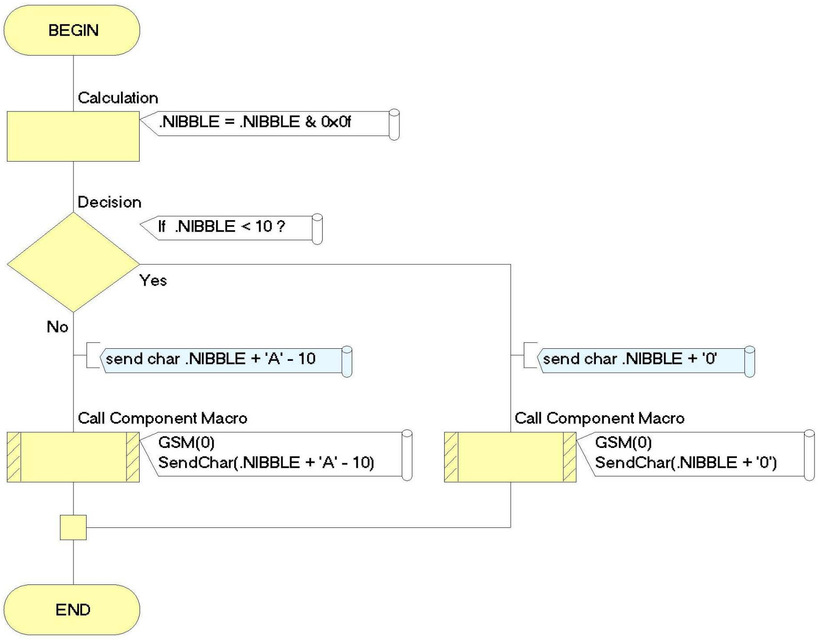


Create a global string **TEXT[255]** to hold the message text.

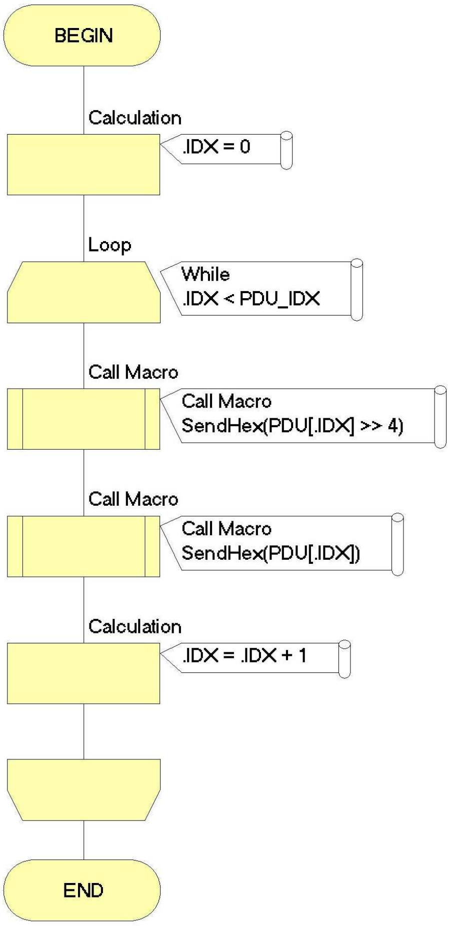
Initialise the counters and index

Loop through the characters of the message and pack them into the PDU buffer.

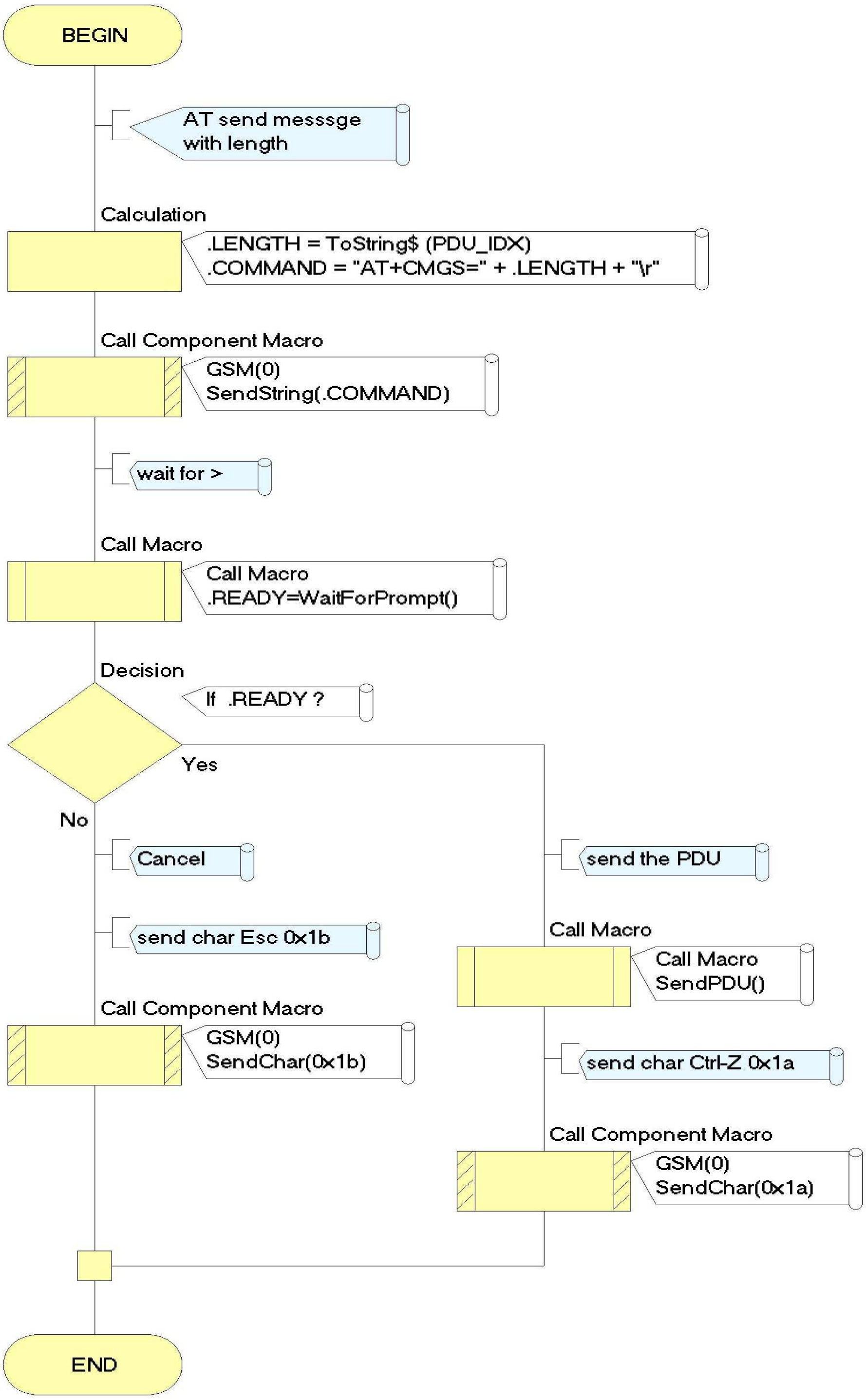
If there are any bits remaining, pack another character to flush them into the PDU buffer.



Create macros **SendHex** and **SendPDU** to send the PDU data to the modem, as it is required to be sent as two hexadecimal character pairs per byte.



Sending the PDU to the GSM Modem



Get the length of the PDU and create the modem command.

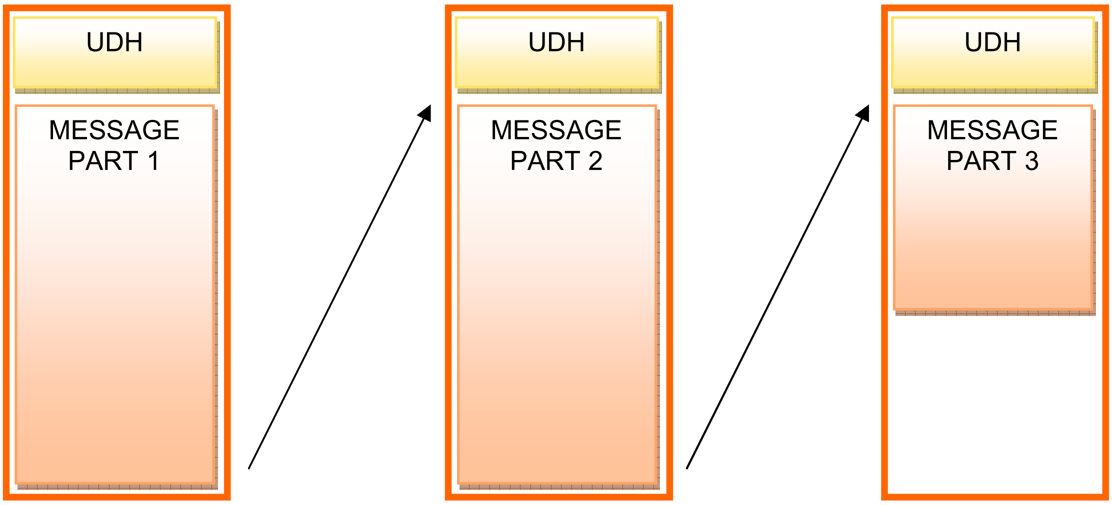
Send the command to the modem.

Wait for the “>” prompt before sending the PDU data to the modem.

Send the PDU data to the modem and terminate with a Ctrl-Z character.

What if I need to send more than 160 characters?

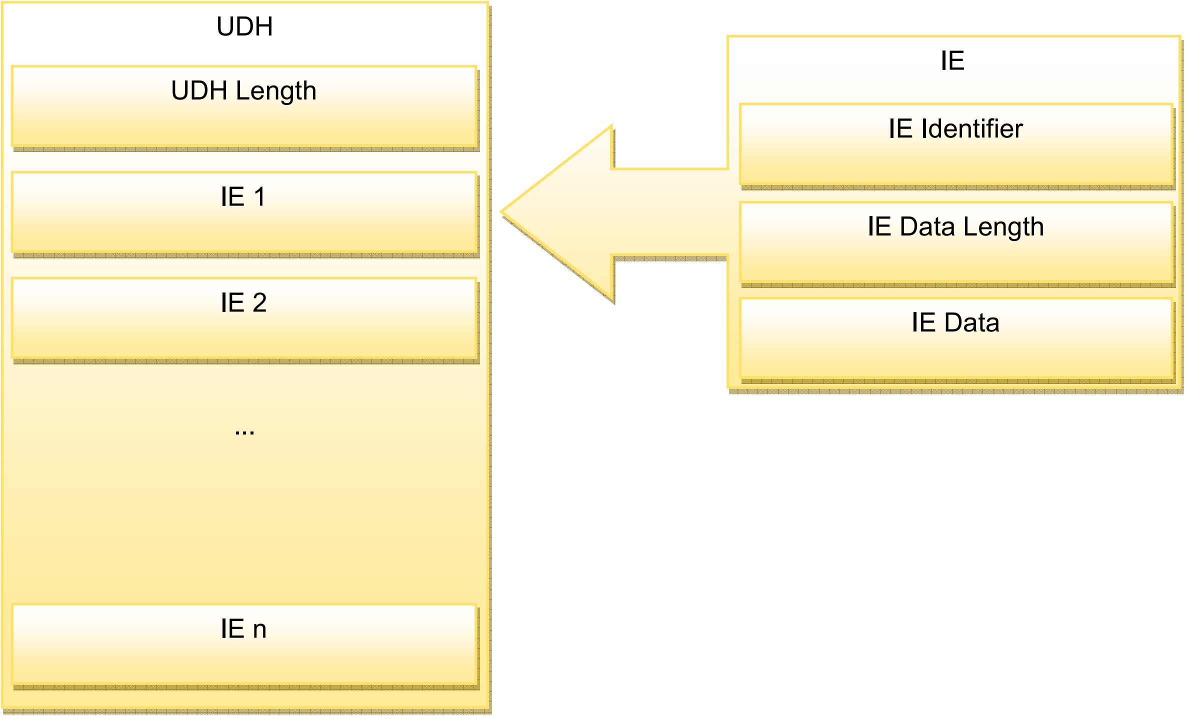
Now that we know how to send messages in PDU mode we can also take advantage of another GSM feature, that being SMS concatenation. Content longer than 140 octets can be sent using multiple SMS messages.



This requires that each SMS message, that forms part of the whole content, has to start with information regarding the segmentation of the content.

This information part of the SMS is contained in a section referred to as the **User Data** Header (UDH) and uses the initial few octets of the SMS payload. Hence this also reduces the available space for message text within each SMS.

Format of the User Data Header (UDH)



The first octet of the payload is the length of the User Data Header and is simply a count of the number of octets of User Data Header that immediately follow, and prior to, the start of the message text. This is followed by any number of Information Elements (IE).
Each Information Element consists of the Information Element Identifier (IEI), one octet, followed by one octet that contains the length of the IE data that follows.

Segmenting the Message

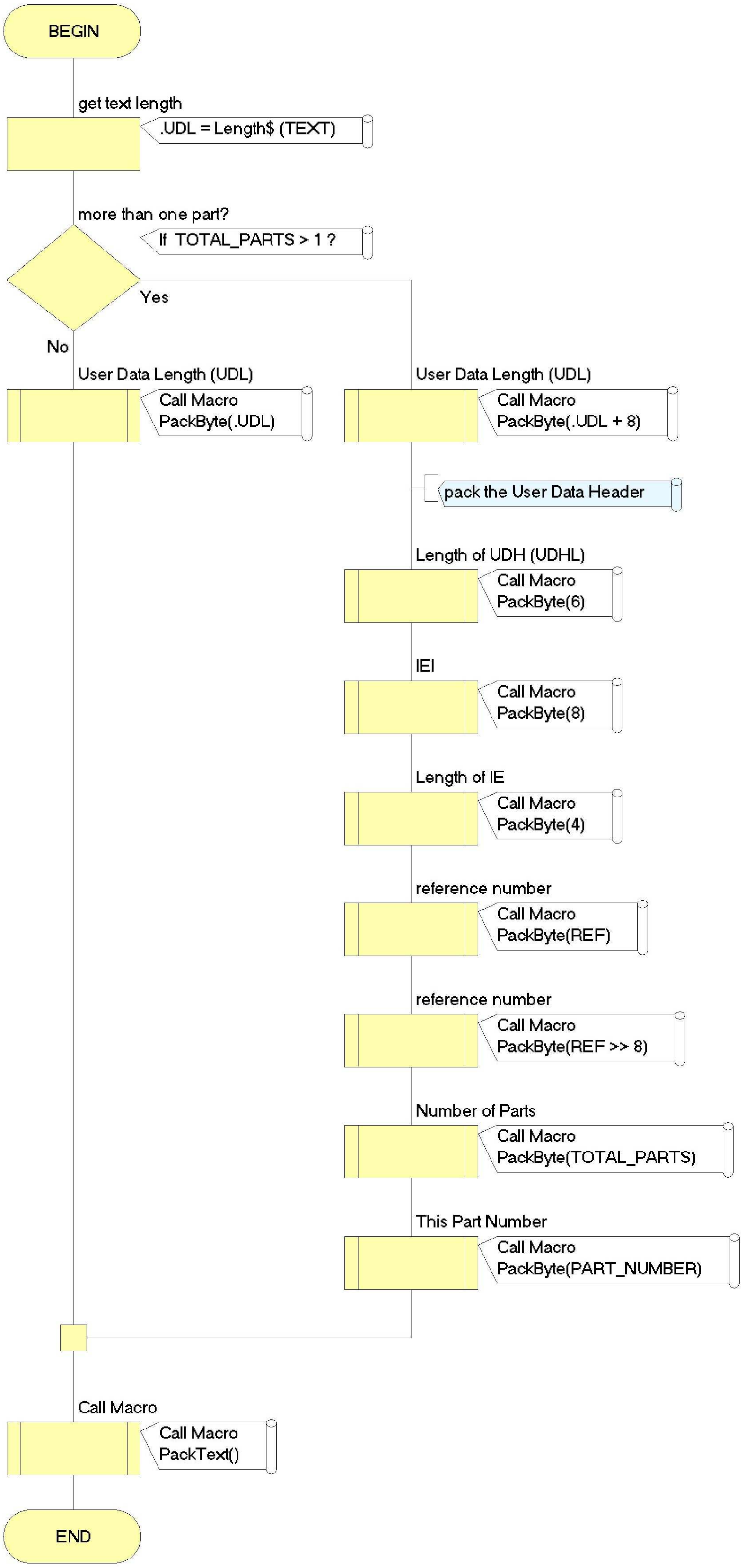
We use the User Data Header to include the segmentation information such that the complete message can be reconstructed at the receiving device.
So for example, if our concatenated message is made up of three parts then each SMS PDU will contain the following octets in the User Data Header at their start:

Index	Content	Description
0	0x06	Length of the User Data Header
1	0x08	IEI for 16 bit concatenation
2	0x04	Length of IE data
3, 4	0x0001	(2 octets) 16 bit reference number for this message. Use the same reference number for all parts belonging to this message, but increment for any subsequent messages.
5	0x03	The total number of parts (segments) that make up this message
6	0x01	The sequence number of this (first) part, increment for subsequent parts, i.e. 2 and 3

The actual message text then follows, remember that we have now used 7 octets of our 140 octet maximum. So there will be 133 octets available for the message text, which will allow the sending of 152 characters.

Also note that if our text is encoded as GSM7 then it must align with our usual septet boundaries. In the case above we have used 7 octets so the text that follows can start immediately in the next octet, otherwise we would need to insert padding bits. This is needed for backwards compatibility with mobile phones that do not understand user data headers, otherwise the whole message would become unreadable garbage rather than just this initial header information.

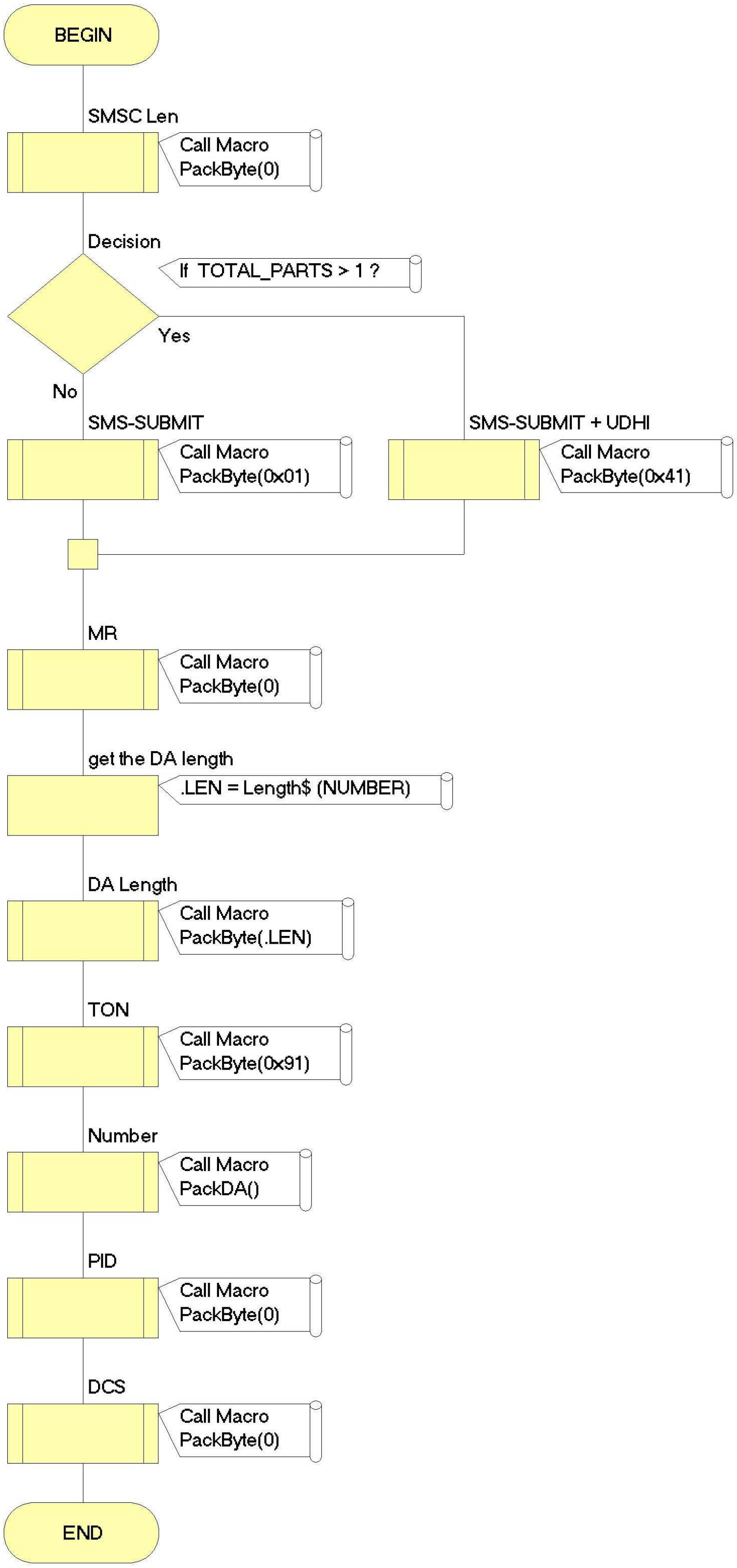
Creating a User Data Header



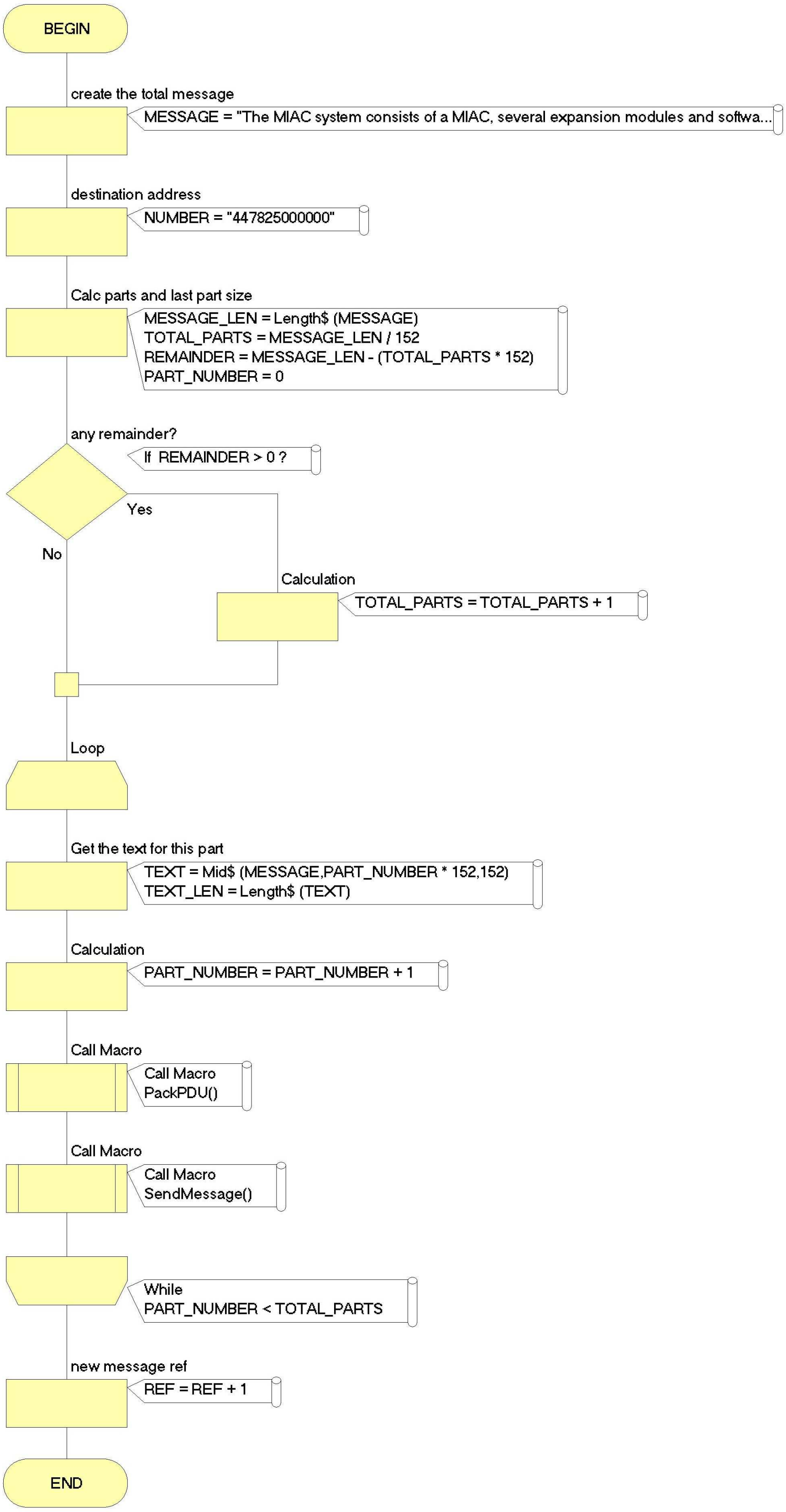
If there is more than one part (segment) to the message then insert the User Data Header information.

Followed by the message text.

The use of a User Data requires the setting of the User Data Header (UDHI) flag to inform the receiving entity that a User Data Header exists and should be processed. This we can do with a small change in the **PackHeader** macro:



The complete message sending macro


BEGIN
create the total message
MESSAGE = "The MIAC system consists of a MIAC, several expansion modules and softwa...
destination address
NUMBER = "447825000000"
Calc parts and last part size
MESSAGE_LEN = Length$ (MESSAGE)
TOTAL_PARTS = MESSAGE_LEN / 152
REMAINDER = MESSAGE_LEN - (TOTAL_PARTS * 152)
PART_NUMBER = 0
any remainder?
If REMAINDER > 0 ?
Yes
No
Calculation
TOTAL_PARTS = TOTAL_PARTS + 1
Loop
Get the text for this part
TEXT = Mid$ (MESSAGE,PART_NUMBER * 152,152)
TEXT_LEN = Length$ (TEXT)
Calculation
PART_NUMBER = PART_NUMBER + 1
Call Macro
Call Macro
PackPDU()
Call Macro
Call Macro
SendMessage()
While
PART_NUMBER < TOTAL_PARTS
new message ref
REF = REF + 1
END

What else can I do with IEIs?

As you will notice, the IEI format is a flexible mechanism whereby handsets can process Information Elements they understand and skip ones that they don't.
The enhanced messaging service EMS is built on this concept and allows the sending of additional audio and visual information in this format within the user data header. See 3GPP TS 23040

Further reading

Below are some links to other resources and articles on related subjects, and technical documentation relating to the hardware used for this project...

E-blocks:	http://www.matrixmultimedia.com/eblocks.php
3GPP TS 23040	http://www.3gpp.org/ftp/Specs/html-info/23040.htm
AT Commands Guide	http://www.telit.com
Learning Centre:	http://www.matrixmultimedia.com/lc_index.php
User Forums:	http://www.matrixmultimedia.com/mmforums
Product Support:	http://www.matrixmultimedia.com/sup_menu.php