GSM Library

Version 3.1

User Manual

DIZZY enterprises

www.dizzy.co.za

Contents

Introduction

GSM modules have made it easy for embedded developers to add hardware capable of connecting to cellular networks to their project.

These GSM modules communicate via simple AT commands. Whilst the AT commands are easy to understand and use, writing reliable embedded code for interfacing with them can be challenging.

We have spent hundreds of hours developing and testing our GSM Library. It is designed to be easy to use, fast, and reliable – with fail-safes and retry mechanisms built in at every step.

Functionality provided by the library includes:

Base Library

- ✓ Read/write the GSM modules RTCC (real-time calendar and clock).
  Since most GSM modules have a facility for connecting a small backup battery to maintain the RTCC, this can be a convenient feature for keeping track of the date and time in the event of power loss.
- ✓ Receive the date/time from the GSM network.
  Most GSM networks can send the current date/time to the GSM module upon registration on the network. The module can use this information to set its own internal RTCC. This feature is currently available for SIMCom, Telit and Quectel modules.
- ✓ Read the modules unique IMEI code.
  The IMEI can be used to uniquely identify the unit.
- ✓ Enter SIM card PIN code (optional).
- ✓ Detect a missed call (including the caller ID).

Text Messaging (SMS) Extension

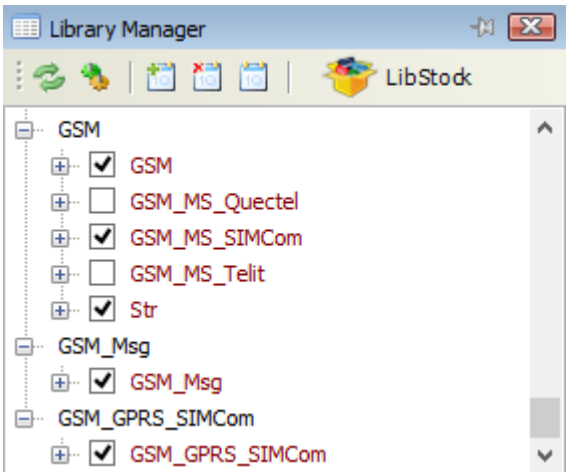
- ✓ Receive text messages.
- ✓ Send text messages.

GPRS Extension

- ✓ HTTP GET.
  This feature is currently only available for SIMCom modules.
- ✓ HTTP POST.
  This feature is currently only available for SIMCom modules.

Quick Start

The GSM Library (and extensions) are provided as mikroElektronika compiler library packages. Assuming that the relevant mikroElektronika compiler is already installed, the library can be easily added to it by using the Package Manager (http://www.mikroe.com/package-manager/). Once installed, the library (and extensions, if applicable) will be listed within Library Manager in the compiler.



GSM Library requires the following in order to operate:

1. Select/deselect the relevant library components in the Library Manager.

GSM	Required	Main GSM Library component.
Str	Required	GSM Library string support functions.
GSM_MS_x	Optional / required for some modules	Manufacturer specific code (select appropriate GSM module manufacturer). Required for Telit modules.
GSM_Msg	Optional	Text message (SMS) functionality.
GSM_GPRS_x	Optional	GPRS functionality.

2. Declare alias for the GSM modules Power_Key (Reset on Telit modules) as well as Status pins.

mikroC

```
sbit GSM_Pwr_Key at LATE2_bit;
sbit GSM_Pwr_Key_Dir at TRISE2_bit;
sbit GSM_Stat at PORTE.B0;
sbit GSM_Stat_Dir at TRISE0_bit;
```

mikroBasic

```
dim GSM_Pwr_Key as sbit at LATE2_bit
dim GSM_Pwr_Key_Dir as sbit at TRISE2_bit
dim GSM_Stat as sbit at RE0_bit
dim GSM_Stat_Dir as sbit at TRISE0_bit
```

mikroPascal

```
var GSM_Pwr_Key : sbit at LATE2_bit;
var GSM_Pwr_Key_Dir : sbit at TRISE2_bit;
var GSM_Stat : sbit at RE0_bit;
```

```
var GSM_Stat_Dir : sbit at TRISE0_bit;
```

Note
For microcontrollers which do not have "TRIS" bits (such as ARM MCUs) the "`GSM_Pwr_Key_Dir`" and "`GSM_Stat_Dir`" declarations should be omitted, and input/output direction setup of the pin should be performed manually in your software. For example, declarations in mikroC Pro for ARM might be:

```
sbit GSM_Pwr_Key at ODR5_GPIOB_ODR_bit;
sbit GSM_Stat at IDR0_GPIOA_IDR_bit;
```

3. Call initialisation routines at startup, and set I/O dirs for the Power_Key and Status pins.

mikroC
```
GSM_Pwr_Key_Dir = 0;
GSM_Stat_Dir = 1;
UART1_Init(9600); // GSM Lib is designed to work at 9600 baud
gsmInit();
gsm_MS_Init();
gsm_Msg_Init(); // Only if GSM_Msg extension is available
gsm_GPRS_Init(); // Only if GSM_GPRS extension is available
```

mikroBasic
```
GSM_Pwr_Key_Dir = 0
GSM_Stat_Dir = 1
UART1_Init(9600) ' GSM Lib is designed to work at 9600 baud
gsmInit()
gsm_MS_Init()
gsm_Msg_Init() ' Only if GSM_Msg extension is available
gsm_GPRS_Init() ' Only if GSM_GPRS extension is available
```

mikroPascal
```
GSM_Pwr_Key_Dir := 0;
GSM_Stat_Dir := 1;
UART1_Init(9600); // GSM Lib is designed to work at 9600 baud
gsmInit();
gsm_MS_Init();
gsm_Msg_Init(); // Only if GSM_Msg extension is available
gsm_GPRS_Init(); // Only if GSM_GPRS extension is available
```

4. Set up a 1ms timer interrupt, and call gsm1msPing() from within the interrupt. Timer Calculator (www.mikroe.com/timer-calculator) can be very helpful for setting up the interrupt.

mikroC
```
gsm1msPing(); // Call from within 1ms timer interrupt
```

mikroBasic
```
gsm1msPing() ' Call from within 1ms timer interrupt
```

mikroPascal
```
gsm1msPing(); // Call from within 1ms timer interrupt
```

5. Call gsmPoll() from within your main program loop as often as possible.

mikroC
```
gsmPoll(); // Call from within 1ms timer interrupt
```

mikroBasic
```
gsmPoll() ' Call from within 1ms timer interrupt
```

mikroPascal
```
gsmPoll(); // Call from within 1ms timer interrupt
```

6. Create a routine called gsmEvent() to handle events from the GSM Library.

mikroC

```
void gsmEvent(char GsmEventType) {
 // Handle GSM Library events here
}
```

mikroBasic

```
sub procedure gsmEvent(dim GsmEventType as char)
  ' Handle GSM Library events here
end sub
```

mikroPascal

```
procedure gsmEvent(GsmEventType : char);
begin
  // Handle GSM Library events here
end;
```

That's it! The basic program structure should now be something like the following:

mikroC

```
sbit GSM_Pwr_Key at LATE0_bit;
sbit GSM_Pwr_Key_Dir at TRISE0_bit;
sbit GSM_Stat at PORTE.B2;
sbit GSM_Stat_Dir at TRISE2_bit;

void interrupt() {
  // <Timer0 Interrupt>
  if (T0IF_bit) { // 1ms Interrupt
    TMR0H = 0xD1;
    TMR0L = 0x20;
    gsm1msPing();
    T0IF_bit = 0; // Clear interrupt flag
  }
  // </Timer0 Interrupt>
}

void gsmEvent(char GsmEventType) {
  // Handle GSM Library events here
}

void main() {
  GSM_Pwr_Key_Dir = 0;
  GSM_Stat_Dir = 1;
  UART1_Init(9600); // GSM Lib is designed to work at 9600 baud
  gsmInit();
  gsm_MS_Init();
  gsm_Msg_Init(); // Only if GSM_Msg extension is available
  gsm_GPRS_Init(); // Only if GSM_GPRS extension is available
  T0CON = 0b10001000; // Set up interrupt timer
  GIE_bit = 1; // Global interrupt enable
  T0IE_bit = 1; // Enable Timer0 overflow interrupts
  while (1) {
    gsmPoll();
    // Add your code here
  }
}
```

mikroBasic

```
dim GSM_Pwr_Key as sbit at LATE2_bit
dim GSM_Pwr_Key_Dir as sbit at TRISE2_bit
dim GSM_Stat as sbit at RE0_bit
dim GSM_Stat_Dir as sbit at TRISE0_bit

sub procedure interrupt()
  ' <Timer0 Interrupt>
  if (T0IF_bit) then ' 1ms Interrupt
    TMR0H = 0xD1
    TMR0L = 0x20
    gsm1msPing()
    T0IF_bit = 0 ' Clear interrupt flag
  end if
  ' </Timer0 Interrupt>
end sub

sub procedure gsmEvent(dim GsmEventType as char)
  ' Handle GSM Library events here
end sub

main:
  GSM_Pwr_Key_Dir = 0
  GSM_Stat_Dir = 1
  UART1_Init(9600) ' GSM Lib is designed to work at 9600 baud
  gsmInit()
  gsm_MS_Init()
  gsm_Msg_Init() ' Only if GSM_Msg extension is available
  gsm_GPRS_Init() ' Only if GSM_GPRS extension is available
  T0CON = %10001000 ' Set up interrupt timer
  GIE_bit = 1 ' Global interrupt enable
  T0IE_bit = 1 ' Enable Timer0 overflow interrupts
  while true
    gsmPoll()
    ' Add your code here
  wend
end.
```

mikroPascal

```
var GSM_Pwr_Key : sbit at LATE2_bit;
var GSM_Pwr_Key_Dir : sbit at TRISE2_bit;
var GSM_Stat : sbit at RE0_bit;
var GSM_Stat_Dir : sbit at TRISE0_bit;

procedure interrupt();
begin
  // <Timer0 Interrupt>
  if (T0IF_bit) then begin  // 1ms Interrupt
    TMR0H := 0xD1;
    TMR0L := 0x20;
    gsm1msPing();
    T0IF_bit := 0; // Clear interrupt flag
  end;
  // </Timer0 Interrupt>
end;

procedure gsmEvent(GsmEventType : char);
```

```
begin
  // Handle GSM Library events here
end;

procedure main();
begin
  GSM_Pwr_Key_Dir := 0;
  GSM_Stat_Dir := 1;
  UART1_Init(9600); // GSM Lib is designed to work at 9600 baud
  gsmInit();
  gsm_MS_Init();
  gsm_Msg_Init(); // Only if GSM_Msg extension is available
  gsm_GPRS_Init(); // Only if GSM_GPRS extension is available
  T0CON := %10001000; // Set up interrupt timer
  GIE_bit := 1; // Global interrupt enable
  T0IE_bit := 1; // Enable Timer0 overflow interrupts
  while (1) do begin
    gsmPoll();
    // Add your code here
  end;
end.
```

Routines and Events

The Event Mechanism

GSM Library uses the gsmEvent() routine to both report and request information. The GsmEventType parameter contains the event type (please see the sections below for details on the various types of events). There are 3 global variables which are used to pass data back and forth with the GSM Library:

Variable	Type
pstrGsmEventData	Pointer to a string (char)
pstrGsmEventOriginatorID	Pointer to a string (char)
dtmGsmEvent	DateTime Structure (more information below)

The use of these variables is event-specific (details in the sections below). Not all of the variables are used by all of the events, and some events may use no variables at all.

dtmGsmEvent consists of the following structure:

mikroC

```
typedef struct DateTime {
  char Year, Month, Day, Hour, Minute, Second;
} TDateTime;
```

mikroBasic

```
structure DateTime
  dim Year, Month, Day, Hour, Minute, Second as char
end structure
typedef TDateTime as DateTime
```

mikroPascal

```
type TDateTime = record
```

```
  Year, Month, Day, Hour, Minute, Second : char;
end;
```

If you are not familiar with structures, then please see the mikroC / mikroBasic / mikroPascal help file for detailed information.

For mikroBasic / mikroPascal users not familiar with pointers, these can be thought of as function parameters passed "by reference" (byref). Pointers can in fact be passed to a function which requires parameters by reference, and this can be an easy way to "work around" the pointer:

mikroBasic

```
sub procedure gsmMissedCall(dim byref OriginatorID as string)
  ' OriginatorID is a string, and pointers can be forgotten about :)
  LCD_Cmd(_LCD_Clear)
  LCD_Out(1,1,OriginatorID)
  LCD_Out(2,1,"(Missed call)")
end sub

sub procedure gsmEvent(dim GsmEventType as char)
  select case GsmEventType
    case gsmevntMissedCall
      ' pstrGsmEventOriginatorID is a string pointer
      ' pstrGsmEventOriginatorID^ (carat symbol at end) defers
      ' to the string pointed to by pstrGsmEventOriginatorID
      ' See the mikroBasic help file for more information
      gsmMissedCall(pstrGsmEventOriginatorID)
  end select
end sub
```

mikroPascal

```
procedure gsmMissedCall(var OriginatorID : string);
begin
  //OriginatorID is a string, and pointers can be forgotten about :)
  LCD_Cmd(_LCD_Clear);
  LCD_Out(1,1,OriginatorID);
  LCD_Out(2,1,'(Missed call)');
end;

procedure gsmEvent(GsmEventType : char);
begin
  case (GsmEventType) of
    gsmevntMissedCall: begin
      // pstrGsmEventOriginatorID is a string pointer
      // pstrGsmEventOriginatorID^ (carat symbol at end) defers
      // to the string pointed to by pstrGsmEventOriginatorID
      // See the mikroBasic help file for more information
      gsmMissedCall(pstrGsmEventOriginatorID);
  end;
end;
```

Power

The gsmPowerSetOnOff routine can be used to instruct the GSM Library to power the GSM module on or off (GSM Library will power the module on by default).

Routines

mikroC
`void gsmPowerSetOnOff(char power_on)`
mikroBasic
`sub procedure gsmPowerSetOnOff(dim power_on as char)`
mikroPascal
`procedure gsmPowerSetOnOff(power_on : char);`

Date/Time (RTCC)

The date/time routines can be used to read from / write to the GSM modules internal RTCC (real-time calendar and clock). Since most GSM modules have a facility for connecting a small backup battery to maintain the RTCC, this can be a convenient feature for keeping track of the date and time in the event of power loss. Additionally, GSM Library will automatically try to obtain the date/time from the GSM network – if it does, then the GSM module will set its RTCC using the data obtained from the network.

Tip
When the event gsmevntMsgRcvd occurs (please see the Text Messaging section for more information), the gsmMsgJustArrived() routine can be used to determine if the text message has just arrived (the message could also possibly be an old one retrieved from the SIM card memory, in which case gsmMsgJustArrived will return false/0). If a text message has just arrived, then its date/time stamp are probably current, and can be used to set the GSM modules RTCC (this must be initiated manually using gsmDateTimeWrite()).

Routines

mikroC
`void gsmDateTimeRead()`
`void gsmDateTimeWrite()`
mikroBasic
`sub procedure gsmDateTimeRead()`
`sub procedure gsmDateTimeWrite()`
mikroPascal
`procedure gsmDateTimeRead();`
`procedure gsmDateTimeWrite();`

Events

gsmevntDateTimeRead

This event occurs after a read of the GSM modules RTCC, initiated by the gsmDateTimeRead() routine, is completed. dtmGsmEvent contains the date/time read.

gsmevntDateTimeWrite

This event occurs just before a write to the GSM modules RTCC, initiated by the gsmDateTimeWrite() routine. dtmGsmEvent should be loaded with the date/time to be written.

IMEI Code

GSM library will automatically read the modules IMEI code. This can be used as a unique identifier.

Events

gsmevntIMEI_Read

pstrGsmEventData points to the read IMEI.

PIN Code

GSM library can optionally enter a PIN code for the SIM card.

Events

gsmevntPIN_Request

pstrGsmEventData should be pointed to the PIN code. If the event is ignored or pstrGsmEventData is pointed to a blank string, then a PIN code will not be entered.

gsmevntPIN_Fail

The entered pin code was incorrect.

Missed Call

Events

gsmevntMissedCall

pstrGsmEventOriginatorID points to a string containing the Caller ID from which the missed call originated.

Network Registration

The gsmReady() routine can be used to determine if the GSM module is registered on the GSM network.

Routines

mikroC
`void gsmReady()`
mikroBasic
`sub procedure gsmReady()`
mikroPascal
`procedure gsmReady();`

Text Messaging

Text Messaging routines and events require the GSM_Msg extension.

Warning
GSM Library uses the SIM card memory for drafting and reading messages. This is done in order to maximise reliability. If the SIM card message memory is not currently empty, then any messages stored on the SIM card will be processed and deleted, and any drafts saved on the SIM card will be sent. Please check that this will not cause any problems before using GSM Library with your SIM card. (Perhaps copy any relevant messages from SIM card memory to phone memory.)

gsmMsgSend() can be used to send a text message, gsmMsgSendPending() can be used to determine if the operation has been processed yet and gsmMsgSendCancel() can be used to cancel the operation.

Messages are first drafted into the SIM card memory before being sent. This is done in order to maximise reliability, and also allows multiple messages to be "qued up" in the SIM card memory before being sent. Once gsmMsgSend() has been called, then the message and DestinationID parameters should not be changed until the message has been drafted (gsmevntMsgDrafted event, gsmMsgSendPending will return false/0), as these parameters are passed as pointers ("by reference").

gsmMsgJustArrived() can be called during a gsmevntMsgRcvd event in order to determine if the message just arrived. If gsmMsgJustArrived() returns false, then the message was read from the SIM card memory and could possibly be old (but not definitely).

Routines

mikroC
`void gsmMsgSend(char *Message, char *DestinationID)`
`char gsmMsgSendPending()`
`void gsmMsgSendCancel()`
`char gsmMsgJustArrived()`
mikroBasic
`sub procedure gsmMsgSend(dim byref Message as string, dim byref DestinationID as string)`
`sub function gsmMsgSendPending() as char`
`sub procedure gsmMsgSendCancel()`
`sub function gsmMsgJustArrived() as char`
mikroPascal
`procedure gsmMsgSend(var Message : string; var DestinationID : string);`
`function gsmMsgSendPending() : char;`
`procedure gsmMsgSendCancel();`
`function gsmMsgJustArrived() : char;`

Events

gsmevntMsgRcvd

Occurs when a GSM message is received. pstrGsmEventData points to a string containing the message text. pstrGsmEventOriginatorID points to a string containing the Caller ID from which the message originated.

gsmevntMsgDrafted

Occurs after a message to be sent has been drafted to the SIM card memory (initiated by gsmMsgSend()). pstrGsmEventData points to a string containing the message ID.

gsmevntMsgDiscarded

Occurs if drafting of a message to the SIM card memory (initiated by gsmMsgSend()) has failed. This could possibly occur if the SIM card memory is full.

gsmevntMsgSent

Occurs when sending of a message is completed. pstrGsmEventData points to a string containing the message ID.

gsmevntMsgSendFailed

Occurs if sending of a message has failed. This could possibly occur if there is no credit/"airtime" loaded on the SIM card. pstrGsmEventData points to a string containing the message ID.

GPRS

GPRS routines and events require the GSM_GPRS extension. Currently, only HTTP GET and POST operations are supported, and only for SIMCom modules.

gsmGprsHttpGet() and gsmGprsHttpPost() can be used to initiate an HTTP GET or POST operation respectively. The result is returned via one or more gsmevntGprsHttpResponseLine events (one event for each line of the response). If the HTTP server returns a result code other than 200 (e.g. "404 – Page not found") then the gsmevntGprsHttpResultErr event will occur. If the GPRS operation fails for any other reason (e.g. no signal), then the gsmevntGprsFailed event will occur.

Routines

mikroC
`void gsmGprsHttpGet(char* url)`
`void gsmGprsHttpPost(char* url, char* postdata)`
`char gsmGprsPending()`
`void gsmGprsCancel()`
mikroBasic
`sub procedure gsmGprsHttpGet(dim byref url as string)`
`sub procedure gsmGprsHttpPost(dim byref url as string,` `dim byref postdata as string)`
`sub function gsmGprsPending() as char`
`sub procedure gsmGprsCancel()`
mikroPascal
`procedure gsmGprsHttpGet(var url : string);`
`procedure gsmGprsHttpPost(var url : string, var postdata : string);`
`function gsmGprsPending() : char;`
`procedure gsmGprsCancel();`

Events

gsmevntGprsHttpResponseLine

This event will occur for each line of the HTTP response received (request initiated by gsmGprsHttpGet() or gsmGprsHttpPost()). pstrGsmEventData points to a string containing the line of data.

gsmevntGprsHttpResultErr

Occurs if the HTTP server returned a result code other than 200 (e.g. "404 – Page not found") as a result of an HTTP GET or POST operation (initiated by gsmGprsHttpGet() or

gsmGprsHttpPost()). pstrGsmEventData points to a string containing the result code returned from the HTTP server.

gsmevntGprsFailed

Occurs if the GPRS operation failed (e.g. no signal).

Tips

Delays

gsmPoll() should be called as often as possible. This means that the use of delay_ms() (or similar) in your code should be avoided. There are however ways to work around this:

Example 1 (Recommended)

Use a timer variable, incremented during the 1ms timer interrupt, to time the delay.

mikroC

```
sbit GSM_Pwr_Key at LATE0_bit;
sbit GSM_Pwr_Key_Dir at TRISE0_bit;
sbit GSM_Stat at PORTE.B2;
sbit GSM_Stat_Dir at TRISE2_bit;

unsigned int wrdDelayTmr;

void interrupt() {
  // <Timer0 Interrupt>
  if (T0IF_bit) { // 1ms Interrupt
    TMR0H = 0xD1;
    TMR0L = 0x20;
    gsm1msPing();
    wrdDelayTmr++;
    T0IF_bit = 0; // Clear interrupt flag
  }
  // </Timer0 Interrupt>
}

void do_delay(unsigned int delaytime_ms) {
  // Delay for a certain amount of time,
  // whilst still allowing the GSM Library to operate
  wrdDelayTmr = 0;
  while (wrdDelayTmr < delaytime_ms) {
    gsmPoll();
  }
}

void gsmEvent(char GsmEventType) {
  // Handle GSM Library events here
}

void main() {
  GSM_Pwr_Key_Dir = 0;
  GSM_Stat_Dir = 1;
```

```
  UART1_Init(9600); // GSM Lib is designed to work at 9600 baud
  gsmInit();
  gsm_MS_Init();
  gsm_Msg_Init(); // Only if GSM_Msg extension is available
  gsm_GPRS_Init(); // Only if GSM_GPRS extension is available
  T0CON = 0b10001000; // Set up interrupt timer
  GIE_bit = 1; // Global interrupt enable
  T0IE_bit = 1; // Enable Timer0 overflow interrupts
  while (1) {
    gsmPoll();
    // Add your code here
  }
}
```

mikroBasic

```
dim GSM_Pwr_Key as sbit at LATE2_bit
dim GSM_Pwr_Key_Dir as sbit at TRISE2_bit
dim GSM_Stat as sbit at RE0_bit
dim GSM_Stat_Dir as sbit at TRISE0_bit

dim wrdDelayTmr as word

sub procedure interrupt()
  ' <Timer0 Interrupt>
  if (T0IF_bit) then ' 1ms Interrupt
    TMR0H = 0xD1
    TMR0L = 0x20
    gsm1msPing()
    Inc(wrdDelayTmr)
    T0IF_bit = 0 ' Clear interrupt flag
  end if
  ' </Timer0 Interrupt>
end sub

sub procedure do_delay(dim delaytime_ms as word)
  ' Delay for a certain amount of time,
  ' whilst still allowing the GSM Library to operate
  wrdDelayTmr = 0
  while (wrdDelayTmr < delaytime_ms)
    gsmPoll()
  wend
end sub

sub procedure gsmEvent(dim GsmEventType as char)
  ' Handle GSM Library events here
end sub

main:
  GSM_Pwr_Key_Dir = 0
  GSM_Stat_Dir = 1
  UART1_Init(9600) ' GSM Lib is designed to work at 9600 baud
  gsmInit()
  gsm_MS_Init()
  gsm_Msg_Init() ' Only if GSM_Msg extension is available
  gsm_GPRS_Init() ' Only if GSM_GPRS extension is available
  T0CON = %10001000 ' Set up interrupt timer
```

```
  GIE_bit = 1 ' Global interrupt enable
  T0IE_bit = 1 ' Enable Timer0 overflow interrupts
  while true
    gsmPoll()
    ' Add your code here
  wend
end.
```

mikroPascal

```
var GSM_Pwr_Key : sbit at LATE2_bit;
var GSM_Pwr_Key_Dir : sbit at TRISE2_bit;
var GSM_Stat : sbit at RE0_bit;
var GSM_Stat_Dir : sbit at TRISE0_bit;

var wrdDelayTmr : word;

procedure interrupt();
begin
  // <Timer0 Interrupt>
  if (T0IF_bit) then begin  // 1ms Interrupt
    TMR0H := 0xD1;
    TMR0L := 0x20;
    gsm1msPing();
    Inc(wrdDelayTmr);
    T0IF_bit := 0; // Clear interrupt flag
  end;
  // </Timer0 Interrupt>
end;

procedure do_delay(delaytime_ms : word);
begin
  // Delay for a certain amount of time,
  // whilst still allowing the GSM Library to operate
  wrdDelayTmr := 0;
  while (wrdDelayTmr < delaytime_ms) do begin
    gsmPoll();
  end;
end;

procedure gsmEvent(GsmEventType : char);
begin
  // Handle GSM Library events here
end;

procedure main();
begin
  GSM_Pwr_Key_Dir = 0;
  GSM_Stat_Dir = 1;
  UART1_Init(9600); // GSM Lib is designed to work at 9600 baud
  gsmInit();
  gsm_MS_Init();
  gsm_Msg_Init(); // Only if GSM_Msg extension is available
  gsm_GPRS_Init(); // Only if GSM_GPRS extension is available
  T0CON := %10001000; // Set up interrupt timer
  GIE_bit := 1; // Global interrupt enable
  T0IE_bit := 1; // Enable Timer0 overflow interrupts
```

```
  while (1) do begin
    gsmPoll();
    // Add your code here
  end;
end.
```

Example 2

Break the delay up into smaller pieces, calling gsmPoll() between each piece. This could result in the delay being slightly longer than desired.

mikroC

```
void do_delay(unsigned int delaytime_ms) {
  // Delay for a certain amount of time,
  // whilst still allowing the GSM Library to operate
  unsigned int ctr;
  for (ctr=0;ctr<delaytime_ms;ctr++) {
    gsmPoll();
    delay_ms(1);
  }
}
```

mikroBasic

```
sub procedure do_delay(dim delaytime_ms as word)
  ' Delay for a certain amount of time,
  ' whilst still allowing the GSM Library to operate
  dim ctr as word
  for ctr = 0 to delaytime_ms
    gsmPoll()
    delay_ms(1)
  next ctr
end sub
```

mikroPascal

```
procedure do_delay(delaytime_ms : word);
begin
  // Delay for a certain amount of time,
  // whilst still allowing the GSM Library to operate
  var ctr : word;
  for ctr := 0 to delaytime_ms do begin
    gsmPoll();
    delay_ms(1);
  end;
end;
```

Multiple UARTs

GSM Library uses the last UART which was initialised or set active (UART_Set_Active(), please see the compiler help file) in order to communicate. If you wish to use multiple UARTS in your project, then you may need to set the UART function pointers to the correct functions before calling gsmPoll() or gsm1msPing():

mikroC

```
UART_Rd_Ptr = &UART1_Read;
UART_Wr_Ptr = &UART1_Write;
UART_Rdy_Ptr = &UART1_Data_Ready;
UART_Tx_Idle_Ptr = &UART1_Tx_Idle;
```

mikroBasic

```
UART_Rd_Ptr = @UART1_Read
UART_Wr_Ptr = @UART1_Write
UART_Rdy_Ptr = @UART1_Data_Ready
UART_Tx_Idle_Ptr = @UART1_Tx_Idle
```

mikroPascal

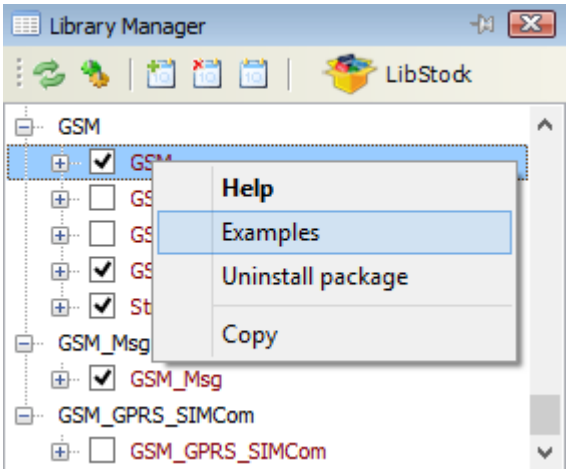
```
UART_Rd_Ptr := @UART1_Read;
UART_Wr_Ptr := @UART1_Write;
UART_Rdy_Ptr := @UART1_Data_Ready;
UART_Tx_Idle_Ptr := @UART1_Tx_Idle;
```

Examples

Various examples of how to use the GSM Library are included with the library package.

Once the package has been installed, these examples can be found in the compilers packages\GSM\Examples folder, or by right-clicking on the GSM Library in Library Manager and then clicking "Examples".

The examples are also downloadable from our website at www.dizzy.co.za.



Disclaimer

This part says that you cannot sue us because we accept no responsibility for any damages whatsoever that may be caused in connection with our products. We've designed them the best we can, but please, use your common sense.