Ultra-Compact Logging with Flash Storage on small MicroMites

Making notes of activity and key information (logging) is an important feature for many systems. Embedded devices are often required to store data for later retrieval and it is vital this data is not lost through power failure etc.

MMBasic provides such a powerful environment it is easy to forget you are working with a single-chip micro-controller, but truth is, there is often not much you can do about storing logs without adding SD Cards or other storage devices - and this can be an unwelcome extra because of program overheads and used pins/board space. Small chips like the three pin UNIO devices can go to about 2KB and WinBond (and others) W25Q chips can add upto 16MB (and higher). This latter is a serious alternative to SD Cards - no big overhead of a file-system or the hardware of card holder, but both options use up pins and may complicate a design - especially a retro-fit.

This article is specifically aimed at logging without additional devices, using the 2KB of flash reserved for storing variables on a MicroMite. Later 'mites provide huge storage areas - the ARMMite gives 128K!

Below is a series of code snippets that show how to use a highly compact, tokenised logging method that can store 270+ lines of log in a string array and save this to flash so it survives power cycles. Unfortunately, you lose the flexibility of free-form text strings; replacing human friendly messages with a single-character token (and in the case below, a two's-compliment single byte number which is enough for storing a simple event counter or temperature in the range of +/- 127 degrees). This does limit the number of messages to about 40 individual message types - you should manage to scrape through ;o)

Firstly here is the Preamble - the bit of code to set everything up. This is all that is required in your start-up section of code. It establishes the limit of the log size (273 lines), dimensions the working array and retrieves any existing log data from flash. In this example, I have grabbed all the flash I can - if you need some for other things or simply don't want a log that big, adjust LogTop down accordingly - that's all.

```
' Preamble
    Option Base 0             ' Operational (live) log starts at element 0
    Const LogTop=272         ' with length 6, we can squeeze 273 (0-272) into
flash - if you are doing one entry per hour that is over 10 days
                   ' but don't save to flash at that rate, a MicroMite
would only last ~2 years! So balance it nicely

    Dim OpLog$(LogTop) Length 6     ' dimension the log array

    Var Restore             ' pull any existing log into the operational
array
```

The following code can be used anywhere in your code as often as you like to make an entry in the operational log.

```
' anywhere in your code... example to log a temperature
    LogEvent LogEnc$("U",GetTemp())    ' to make an entry in the log
```

One problem with the on-chip flash is it has a life-time of about 20,000 write cycles. Please read the

“Micromite Special Features” section of the MicroMite Manual. Bearing this in mind, it is vital to avoid writing the log to flash when it isn't important. I tend to do it only at key points, when I have data I really don't want to lose. In one application, I save the log at dawn and dusk and on startups so 2 writes a day. The chip should last 20+ years. Remember that adding an event to the log doesn't save the log to flash - you have to save the log explicitly. You need to balance the risk of losing data that hasn't been saved versus the cost to your hardware.

```
    Var Save OpLog$()        ' save the array to flash - don't do this too often, try to save
                    ' the log only at key points so as not to wear out the flash
```

This code dumps out the contents of the log. The log is a FIFO list with new entries added to the end after everything else has been shuffled down one position. This ensures I always keep the latest data but older logs will be lost as space is consumed. Note that “” is used to determine an un-used log slot and each used entry is prefixed with “@” to indicate it is historical data.

```
' To dump the log
    For x=0 to LogTop
        If OpLog$(x)<>"" Then Print "@"+LogDec$(OpLog$(x))
    Next
```

This gives stuff like this:

```
@02-04-2018 00:00:00 Temp 9C
@02-04-2018 00:00:00 Sensor triggers in last hour=2
@02-04-2018 01:00:00 Temp 7C
@02-04-2018 02:00:00 Temp 7C
@02-04-2018 02:00:00 Sensor triggers in last hour=11
@02-04-2018 03:00:00 Temp 6C
@02-04-2018 03:00:00 Sensor triggers in last hour=1
@02-04-2018 04:00:00 Temp 6C
@02-04-2018 05:00:00 Temp 7C
@02-04-2018 05:31:00 Dawn. Temp 7C
@02-04-2018 06:00:00 Temp 8C
@02-04-2018 07:00:00 Temp 8C
@02-04-2018 08:00:00 Temp 9C
@02-04-2018 09:00:00 Temp 9C
@02-04-2018 10:00:00 Temp 10C
@02-04-2018 11:00:00 Temp 11C
@02-04-2018 12:00:00 Temp 12C
```

Finally, here are the Subs and Functions that encode and decode the log lines. Note that when encoding a log string, you supply your token e.g. “C”. It is important the decode function knows what a token means to restore the “human readable” nature of the log entry.

```
' The Subs & Functions

' add an item to the event log - only manages and adds the log entry to the array, doesn't save to flash
```

```
    Sub LogEvent(x$)
        Local Integer n
        For n=1 To LogTop
            OpLog$(n-1)=OpLog$(n)
        Next
        OpLog$(LogTop)=x$
    End Sub

' encode a log string
    Function LogEnc$(t$,n As Integer)
        Local Integer m
        Local z$,a$
        a$=Hex$(UnixTime(Now()))
        For m=1 To 8 Step 2
            z$=z$+Chr$(Val("&h"+Mid$(a$,m,2)))
        Next
        LogEnc$=z$+Chr$(n And 255)+Ucase$(t$)
    End Function

' decode a log string
    Function LogDec$(a$)
        Local l$,v$
        Local Integer m
        For m=1 to 4
            l$=l$+Hex$(Asc(Mid$(a$,m,1)),2)
        Next
        l$=HumanTime(Val("&h"+l$))+" "
        v$=Str$(SgnX(7,Asc(Mid$(a$,5))))
        Select Case Mid$(a$,6,1)        ' have whatever you want here in the
case statements
            Case "O"
                LogDec$=l$+"System Started. Watchdog event="+v$
            Case "U"
                LogDec$=l$+"Dawn. Temp "+v$+"C"
            Case "D"
                LogDec$=l$+"Dusk. Temp "+v$+"C"
            Case "T"
                LogDec$=l$+"Temp "+v$+"C"
            Case "S"
                LogDec$=l$+"Sensor triggers in last hour="+v$
            Case Else
                LogDec$=l$+"Unknown"
        End Select
    End Function
```

Dependencies

UnixTime()
HumanTime()
Now()

SgnX()
… and their child dependencies.

The above solution is taken from one highly specific requirement. Yours may vary and you can tweak the code to add more or bigger tokens or more values etc.

See also

MMBasic code_pack to Read and Write Winbond Flash Memories