BIN8.BAS

This module is part of the original MMBasic library. It is reproduced here with kind permission of Hugh Buckle and Geoff Graham. Be aware it may reference functionality which has changed or is deprecated in the latest versions of MMBasic.

Although MMBasic contains a function to return a binary string of any length from an argument, this module is interesting because it contains a number of useful function beyond simple conversion.

BIN8.BAS ReadMe - crackerjack, March 2012

The BIN8.BAS “library” of byte-wide bit manipulation functions makes use of MMBasic 3.2's extremely powerful user-defined function capabilities. The library is 8-bit oriented, but could easily be adapted for 16-bit usage if required.

User-defined functions, combined with MMBasic 3.2's Ctrl-F feature in the full-screen editor enable the programmer to re-use libraries of proven, tested code and together, these features are a boon to the productivity of an MMBasic programmer.

Ctrl-F inserts text from a nominated file at the current line of the cursor in the editor.

It is important to note that the calling convention in MMBasic for both functions and sub's (let's call them collectively “routines”), is ByRef (by reference). This means that the *address* of the actual variable is passed to the routine, and as such, changing the value of the passed-in variable in the routine, will change the value of the variable outside of the routine (i.e. the variable as referenced within the routine, is not a copy of the variable outside of the routine). This can have far-reaching (and sometimes unexpected consequences), so be aware of this and make use of the “Local” statement to define variables with a scope local to the routine into which the passed-in variable's value can be transferred if necessary.

The functions in this library are generally self-explanatory, but each is summarised below along with any tips, tricks or traps. Where applicable, functions call a utility function (CheckByte), which will fail with an Error if the value passed-in is outside the value range of 0 to 255.

GetBit

Allows for determining the value (0 or 1) of any given bit in a byte. The function's return value is the binary value of the bit in the byte. Possibly useful when interfacing to other devices via SPI, I2C, etc.

```
Function GetBit(byte_in, bit)
  ' Return the value (0/1) of a specified bit in a given byte
  CheckByte(byte_in)
  If bit < 0 Or bit > 7 Then Error "Invalid bit value"
  GetBit = (byte_in And (2 ^ bit)) > 0
End Function
```

SetBit

SetBit is not a function, but a user-defined Sub. SetBit does not actually return a value, but instead sets the specified bit in the byte of the provided variable. Possibly useful when interfacing to other devices, etc.

```
Sub SetBit(byte_in, bit)
  ' Sets the given bit of a given byte
  ' Note: **PassByRef** - The passed in variable's value is changed
  CheckByte(byte_in)
  If bit < 0 Or bit > 7 Then Error "Invalid bit value"
  byte_in = byte_in Or (2 ^ bit)
End Sub
```

Invert

Invert sets bits in a byte to their binary opposite (or complement), i.e. if a bit is 0, it is set to 1; if 1, then it is set to 0. The function returns a value with bits inverse to the byte which was passed-in. Possibly useful in ones & twos-complement arithmetic.

```
Function Invert(byte_in)
  'Inverts bits in the byte
  CheckByte(byte_in)
  Invert = byte_in Xor &hFF
End Function
```

ShiftL

Left shifts the bits in the byte by the given number of positions. Bits to the left of bit 7 are discarded (arithmetic shift). The function returns the left-shifted byte value. Useful for a variety of operations.

```
Function ShiftL(byte_in, shift)
  ' Logically shifts bits left by given quantity
  CheckByte(byte_in)
  If shift < 1 Or shift > 7 Then Error "Invalid shift value"
  ShiftL = ((byte_in * 2 ^ shift) And &hFF)
End Function
```

ShiftR

Right shifts the bits in the byte by the given number of positions. Bits to the right of bit 0 (?) are discarded (arithmetic shift). The function returns the right-shifted byte value. Useful for a variety of operations.

```
Function ShiftR(byte_in, shift)
  ' Logically shifts bits right by given quantity
  CheckByte(byte_in)
  If shift < 1 Or shift > 7 Then Error "Invalid shift value"
  ShiftR = (byte_in \ 2 ^ shift)
End Function
```

RotL

Rotates the bits left-wise through the byte by the given number of positions (logical, or circular shift). The function returns the left-rotated byte value.

```
Function RotL(byte_in, rot)
  ' Rotates (circular shift) bits left by given quantity
  CheckByte(byte_in)
  If rot < 1 Or rot > 7 Then Error "Invalid rotate value"
  RotL = ((byte_in * 2 ^ rot) + ((byte_in * 2 ^ rot)\&h100)) And &hFF
End Function
```

RotR

Rotates the bits right-wise through the byte by the given number of positions (logical, or circular shift). The function returns the right-rotated byte value.

```
Function RotR(byte_in, rot)
  ' Rotates (circular shift) bits right by given quantity
  CheckByte(byte_in)
  RotR = RotL(byte_in, (8 - rot))
End Function
```

These two “utility” functions exist:

Bin8$

This function returns a string containing the bit values of the byte passed-in, padded to 8 characters. e.g. the statement

```
Bin8$(2)
```

will print the following line to the display: 00000010 It is similar to the existing Bin$ function in MMBasic but left-padded with zero's.

```
Function Bin8$(byte_in)
 Bin8$ = Right$(String$(8,"0")+Bin$(byte_in),8)
End Function
```

CheckByte

This function raises an error if the value passed-in is outside of the range 0 → 255. While this function is used by a number of functions within the library, it may be useful in it's own right.

```
Sub CheckByte(byte_in)
  If byte_in < 0 Or byte_in > 255 Then Error "Value not in range"
End Sub
```