4. Utility Library

Useful Definitions Used in this Document

Through out this document, the following symbol definitions were assumed.

```
#define BYTE        unsigned char
#define WORD        unsigned short
#define DWORD  unsigned int
#define BOOLEAN     unsigned char
#define FALSE  0
#define TRUE        1
#define NULL        ((void *) 0)
```

4.1 Constant data

4.1.1 const BYTE month[12][3]

Function:
Provide English short form for the 12 months.

4.2 Pack data functions

This pack data functions are built to facilitate pack message. To use it, you must set the buffer start and buffer length using set_pptr. If the data cannot be fitted into the buffer, it will not be packed. Besides, no matter the buffer is in working memory or protected SRAM, it will automatically do it for you.

4.2.1 BYTE * set_pptr(BYTE * ptr, DWORD len)

Function:
Set pack data pointer for other pack functions.

Input:
ptr -> Target location.

Return:
Same as input.

4.2.2 BYTE * get_pptr(void)

Function:
Get pack data pointer.

Input:
None.
Return:
Content of the pack pointer.

4.2.3 DWORD pack_byte(DWORD value)

Function:
Store the data byte into the pointed location.
Increment the pack data pointer by 1.

Input:
Data byte to be saved.

Return:
Same as input.

4.2.4 void pack_nbyte(DWORD value, DWORD len)

Function:
Repeat storing data byte into the pointed location.
Increment the pack data pointer by the number of data stored.

Input:
value = Data byte to be saved.
len = Number of repetition.

Return:
None.

4.2.5 void pack_mem(BYTE * source, DWORD len)

Function:
Store a block of data into the pointed location.
Increment the pack data pointer by the number of data stored.

Input:
source = Location of data to be saved.
len = Length of data.

Return:
None.

4.2.6 void pack_null(DWORD len)

Function:
Repeat storing of 0x00 data into the pointed location.
Increment the pack data pointer by the number of data stored.

Input:
Number of repetition.

Return:
None.

4.2.7 void pack_space(DWORD len)

Function:
Repeat storing of 0x20 data into the pointed location.
Increment the pack data pointer by the number of data stored.

Input:
Number of repetition.

Return:
None.

4.2.8 void pack_zero(DWORD len)

Function:
Repeat storing of 0x30 data into the pointed location.
Increment the pack data pointer by the number of data stored.

Input:
Number of repetition.

Return:
None.

4.2.9 void pack_str(char * string)

Function:
Store a string of data into the pointed location.
Increment the pack data pointer by the number of data stored.

Input:
Location of string to be stored.

Return:
None.

4.2.10 DWORD get_byte(void)

Function:
Retrieve a data byte from the pointed location.
Increment the pack data pointer by 1.

Input:
Return:
Content of data byte at the pointed location.
None.

4.2.11 DWORD peek_byte(void)

Function:
Peek into the data content of the pointed location.

Input:
None.

Return:
Content of data byte at the pointed location.

4.2.12 DWORD inc_pptr(DWORD increment)

Function:
Increment the pack data pointer by the selected value.

Input:
Increment.

Return:
Same as input.

4.2.13 DWORD dec_pptr(DWORD decrement)

Function:
Decrement the pack data pointer by the selected value.

Input:
Decrement.

Return:
Same as input.

4.2.14 DWORD get_distance(void)

Function:
Obtain the distance between the current pointed location and the initial starting point.

Input:
None.

Return:
Distance between current pointed location and the initial starting point.

4.2.15 void pack_hex(DWORD value)

Function:
Convert the input value into 2 bytes Hex-decimal value and store the equivalent ASCII character into the pointed location.
High nibble will be stored first.
Pack pointer will be incremented by 2.

Input:
Data to be stored.

Return:
None.

4.2.16 DWORD get_word()

Function:
Retrieve a data word from the pointed locationin little endian format.
Increment the pack data pointer by 2.

Input:

Return:
Content of data word at the pointed location.
None.

4.2.17 DWORD pack_word(DWORD value)

Function:
Store the data word into the pointed location.
Increment the pack data pointer by 2.

Input:
Data byte to be saved.

Return:
Same as input.

4.2.18 void get_mem(BYTE * dest,DWORD len)

Function:
Read a block of data into the pointed location.
Increment the pack data pointer by the number of data read.

Input:
dest = Location of data to be read
len = Length of data.

Return:
None.

4.2.19 void pack_lf()

Function:
Store the data byte '\n' into the pointed location.
Increment the pack data pointer by 1.

Input:
None

Return:
None

4.2.20 void pack_lfs(DWORD len)

Function:
Repeat storing of '\n' data into the pointed location.
Increment the pack data pointer by the number of data stored.

Input:
Number of repetition.

Return:
None.

4.2.21 void split_data(BYTE * source,DWORD len)

Function:
Split a block of compressed data and pack them into the pointed location.
Increment the pack data pointer by the number of split data.

Input:
source = Location of data to be split and store
len = Length of data.

Return:
None.

4.2.22 void bindec_data(DWORD value,DWORD len)

Function:
Convert a binary dword into a number of ASCII digits and pack them
Increment the pack data pointer by the length of thedigits.

Input:
value = DWORD holds binary value
len = length of ASCII digit to be saved

Return:

None

4.2.23 DWORD buffer_overflow(DWORD len)

Function:
To check if the buffer has enough empty space to whole the length of data as specified

Input:
Length of data

Return:

TRUE	=> has enough space
FALSE	=> not enough space

4.3 Search functions

4.3.1 WORD fndb(void * source, DWORD target, DWORD len)

Function:
Search the target byte within the pointed buffer.

Input:
source = Data buffer to be search.
target = Data byte to look for.
len = Maximum length of search.

Return:
Target offset.
Return len if target no found.

4.3.2 BYTE * fndbptr(BYTE * source, DWORD target, DWORD len)

Function:
Search the target byte within the pointed buffer.

Input:
source = Data buffer to be search.
target = Data byte to look for.
len = Maximum length of search.

Return:
Pointer to target.
Return (source + len) if target no found.

4.3.3 DWORD skpb(BYTE * source, DWORD target, DWORD len)

Function:
Skip the target byte within the pointed buffer.

Input:
source = Data buffer to be search.
target = Data byte to look for.
len = Maximum length of search.

Return:
Offset of first location not equal to target.
Return len if buffer all filled with target.

4.3.4 BYTE * skpbptr(BYTE * source, DWORD target, DWORD len)

Function:
Skip the target byte within the pointed buffer.

Input:
source = Data buffer to be search.
target = Data byte to look for.
len = Maximum length of search.

Return:
Pointer to first location not equal to target.
Return (source + len) if buffer all filled with target.

4.4 Data conversion functions

4.4.1 void compress(BYTE * dest, BYTE * source, DWORD pair)

Function:
Compress the ASCII source buffer to BCD output.

Input:
dest = Data buffer for storing compressed output.
source = Data buffer to be compressed.
pair = Number of data pair for compression.

Return:
None.

4.4.2 void compress_30(BYTE * dest, BYTE * source, DWORD pair)

Function:
Compress the ASCII source buffer to BCD output by just masking the upper nibble.

Input:
dest = Data buffer for storing compressed output.
source = Data buffer to be compressed.
pair = Number of data pair for compression.

Return:
None.

4.4.3 void split(BYTE * dest, BYTE * source, DWORD pair)

Function:
Split the BCD source buffer into ASCII output.

Input:
dest = Data buffer for storing split output.
source = Data buffer to be split.
pair = Number of data pair for decompression.

Return:
None.

4.4.4 void split_30(BYTE * dest, BYTE * source, DWORD pair)

Function:
Split the BCD source buffer by just adding 0x30

Input:
dest = Data buffer for storing split output.
source = Data buffer to be split.
pair = Number of data pair for decompression.

Return:
None.

4.4.5 DWORD hex_digit(DWORD value)

Function:
Convert a 4-bit input to an ASCII HEX digit.

Input:
BCD input (only lower 4 bits are significant).

Return:
BCD value in ASCII representation.

4.4.6 DWORD hex_value(DWORD value)

Function:
Convert an ASCII HEX digit into its equivalent 4 bit binary value.

0x30 - 0x3F	=> 0x00 - 0x0F
‘A’ - ’F’	=> 0x0A - 0x0F
‘a’ - ’f’	=> 0x0A - 0x0F
0x00 - 0x0F	=> 0x00 - 0x0F

Input:
ASCII HEX digit.
Return:
Binary value of the input HEX digit.

4.4.7 DWORD asc2val(BYTE * ptr)

Function:
Convert the pointed 2 decimal digits into its equivalent binary value.
'00' => 0x00
'99' => 0x63

Input:
Decimal digits in ASCII representation.

Return:
Binary value of the input decimal digits.

4.4.8 DWORD dig2val(DWORD w)

Function:
Convert a word of 2 ASCII digits into its equivalent binary value.
'00' => 0x00
'99' => 0x63

Input:
Decimal digits in word.

Return:
Binary value of the input decimal digits.

4.4.9 DWORD val2bcd(DWORD c)

Function:
Convert a binary value from 0 to 99 to a BCD value.

Input:
Binary value in byte.

Return:
BCD value of the input binary value.

4.4.10 DWORD bit_flip(DWORD data_byte)

Function:
Reverse the bit order of the input.
0x01 => 0x80
0x37 => 0xEC

Input:
Byte value.

Return:
Bit flip of the input.

4.4.11 DWORD dec2bin(BYTE * source, DWORD len)

Function:
Convert a string of ASCII digit to a dword binary value.

Input:
Source = ASCII digit pointer
Len = length of the digit string

Return:
Binary value of the input ASCII digits.

4.4.12 DWORD bcd2val(DWORD bcd)

Function:
Convert a BCD byte into its equivalent binary value.
  ‘00’ => 0x00
  ‘99’ => 0x63

Input:
Byte holds BCD digits

Return:
Binary value of the BCD digits.

4.4.13 DWORD bcd2bin(DWORD bcd)

Function:
Convert a BCD word into its equivalent binary value.

Input:
Word holds BCD digits

Return:
Binary value of the BCD digits.

4.4.14 DWORD bin2bcd(DWORD binary)

Function:
Convert a binary word into its equivalent BCD value.

Input:
Word holds binary value

Return:
BCD value of the binary word.

4.4.15 void bin2dec(DWORD w, BYTE * dest, DWORD len)

Function:
Convert a binary dword into a number of ASCII digits

Input:
W = dword holds binary value
Dest = destination to store the ASCII digit
Len = length of ASCII digit to be saved

Return:
None

4.4.16 void lbin2asc(BYTE * dest, unsigned long val)

Function:
Convert an unsigned long into a 10 digits string

Input:
Dest = destination to store the ASCII digit
val = value to be converted

Return:
None

4.4.17 void lbin2bcd(BYTE * dest, unsigned long val)

Function:
Convert an unsigned long into a 6-byte BCD string. The first BCD byte is 0x00.

Input:
Dest = destination to store the BCDdigit
val = value to be converted

Return:
None

4.4.18 void dbin2asc(BYTE * dest, unsigned long long val)

Function:
Convert a double into a 20 digits string

Input:

Dest	= destination to store the ASCII digit
val	= unsigned long long 64-bit value

Return:
None

4.4.19 void dbin2bcd(BYTE * dest, unsigned long long val)

Function:
Convert a double into a 10 byte BCD digits

Input:
Dest = destination to store the BCD digit
val = unsigned long long 64-bit value

Return:
None

4.4.20 DWORD decbin4b(BYTE * source, DWORD len)

Function:
Convert a sting of digit into unsigned long binary value

Input:
source = digit string pointer
len = length of the digit string

Return:
Unsigned long result

4.4.21 unsigned long long decbin8b(BYTE * source, DWORD len)

Function:
Convert a sting of digit into unsigned double value

Input:
source = digit string pointer
len = length of the digit string

Return:
Unsigned long long result

4.4.22 void lbcd2lbin(BYTE *pbcd,DWORD bcd_size,DWORD *pbin)

Function:
Convert a sting of bcd into a string of binary value

Input:
pbcd = bcd string pointer
bcd_size = length of the bcd string
pbin = binary DWORD string pointer to be saved in Big-endian format

Return:
None

4.4.23 DWORD swap_w(DWORD w)

Function:
To swap the high and low byte of the input word

Input:
Word to be swap

Return:
Swapped word.

4.5 Date related functions

4.5.1 DWORD days_of_month(DWORD month)

Function:
Find the maximum number of days in a month.

Input:
Numeric representation of the month.
  0x00 = January
  0x0B = December

Return:
Maximum number of days in the month.
  Feb => 29
  Apr, Jun, Sep, Nov => 30
  others => 31

4.5.2 BOOLEAN date_ok(BYTE * dtg)

Function:
Check if the input is a valid dtg format.
  dtg format: CCYYMMDDHHMMSS
        e.g. 19981231235959

Input:
Pointer to dtg buffer.

Return:
TRUE => valid dtg format.

4.5.3 BOOLEAN leap_year(DWORD year)

Function:
Check if the input year is a leap year.

Input:
Year. (e.g. 1997)

Return:
TRUE => leap year.

4.5.4 BOOLEAN dtg2asc(BYTE * ptr, BYTE * dtg, BOOLEAN century)

Function:
Convert the pointed dtg buffer into ASCII display format.
  dtg format: CCYYMMDDHHMMSS
        e.g. 19981231235959
  display format: DEC 31   23:59:59 without century
            DEC 31,2000 23:59:59 with century

Input:
ptr = Pointer to output buffer
dtg = Pointer to dtg buffer.
Century = include century or not

Return:
TRUE => conversion done.
FALSE => error.

4.6 Data check functions

4.6.1 BOOLEAN parity(DWORD data_byte)

Function:
Find the parity of the input.

Input:
Byte value.
Return:

TRUE	=> input is odd parity. (e.g. 0x31)
FALSE	=> input is even parity. (e.g. 0x30)

4.6.2 BOOLEAN isbdigit(BYTE * ptr, DWORD len)

Function:
Check if contents of the input buffer are in ASCII numeric format.

Input:
ptr = Pointer to input buffer.
len = Length of buffer.

Return:
TRUE => contents are in ASCII numeric format.

4.6.3 DWORD chk_digit(BYTE * ptr, DWORD len)

Function:
Find the check digit of the input buffer.
Contents of input buffer must be in ASCII numeric format.

Input:
ptr = Pointer to input buffer.
len = Length of buffer.

Return:
Check digit in ASCII (‘0’ - ‘9’).

4.6.4 BOOLEAN chk_digit_ok(BYTE * ptr, DWORD len)

Function:
Check if the input buffer has the correct check digit.
Contents of input buffer must be in ASCII numeric format.

Input:
ptr = Pointer to input buffer.
len = Length of buffer.

Return:
TRUE => correct check digit.
FALSE => error.

4.6.5 DWORD crc(BYTE * msg, DWORD len)

Function:
Calculate the CCITT CRC-16 of the input buffer.
Input:
msg = Pointer to input buffer.
len = Length of buffer.

Return:
CCITT CRC-16 of the input buffer.

4.6.6 BYTE cal_crc7(BYTE * msg, DWORD len)

Function:
Calculate the CRC-7 of the input buffer.

Input:
msg = Pointer to input buffer.
len = Length of buffer.

Return:
CRC-7 of the input buffer.

4.6.7 DWORD lrc(BYTE * d_ptr, DWORD len)

Function:
Calculate the LRC of the input buffer.

Input:
d_ptr = pointer to input buffer.
len = Length of buffer.

Return:
LRC of the input buffer.

4.6.8 DWORD checksum(BYTE * d_ptr, DWORD len)

Function:
Calculate the 16-bit checksum of the input buffer by summing of byte in a word.

Input:
d_ptr = pointer to input buffer.
len = Length of buffer.

Return:
16-bit checksum of the input buffer.

4.7 Arithmetic function

4.7.1 DWORD bcdadd(BYTE * dest, BYTE * src, DWORD len)

Function:
To add two packed BCD.

Input:
dest = destination pointer
src = source pointer
len = Length of BCD to be added.

Return:
carry

4.7.2 DWORD bcdsub(BYTE * dest, BYTE * src, DWORD len)

Function:
To do BCD subtraction of two packed BCD.

Input:
dest = destination pointer
src = source pointer
len = Length of BCD to be subtracted.

Return:
borrow

4.7.3 void bcdinc(BYTE * dest, DWORD len)

Function:
To increment a packed BCD.

Input:
dest = destination pointer
len = Length of BCD to be incremented.

Return:
none

4.7.4 DWORD ascadd(BYTE * dest, BYTE * src, DWORD len)

Function:
To add two ASCII numbers.

Input:
dest = destination pointer
src = source pointer
len = Length of ASCII number to be added.
Return:
carry

4.7.5 DWORD ascsub(BYTE * dest, BYTE * src, DWORD len)

Function:
To do ASCII number subtraction.

Input:
dest = destination pointer
src = source pointer
len = Length of ASCII number to be subtracted.

Return:
borrow

4.7.6 void ascinc(BYTE * dest, DWORD len)

Function:
To increment ASCII number

Input:
dest = destination pointer
len = Length of ASCII number to be incremented.

Return:
none

4.8 Miscellaneous functions

4.8.1 BYTE * scanbyte(BYTE * source, DWORD * result)

Function:
To convert the ASCII numeric input buffer to its equivalent binary value, it will stop at non-numeric input.

Input:

source	= Pointer to input buffer to be scanned.
result	= Pointer to BYTE for storing the scanned result.

Return:
Pointer to first location that is not numeric.

4.8.2 BYTE * amount_conv(BYTE * output, BYTE * source, DWORD decimal)

Function:
Convert the input buffer to amount display or edit format.

```
   e.g. source -> ”HKD $.5”
          if decimal = 0x02, output -> “HKD $0.50”
          if decimal = 0x12, output -> “50”
          if decimal = 0x00, output -> “HKD $0”
          if decimal = 0x10, output -> “”
   e.g. source -> ”HKD $123.456”
          if decimal = 0x02, output -> “HKD $123.45”
          if decimal = 0x12, output -> “12345”
```

Input:
output = Pointer to output string.
source = Pointer to input string.
decimal = Format control.

```
                Bit 3:0 - Number of decimal place in effect.
                Bit 7:4 - Output format control.
                                0 => Copy source string format to output.
                                1 => Minimum numeric representation.
```

Return:
Pointer to point of termination in source input string.

4.8.3 void justify_right(BYTE * source, DWORD filler, DWORD len)

Function:
To right justify the input string.
Output will not be NULL terminated.
e.g. source -> “123”
if (len = 8, filler = ‘0’), source -> ‘00000123’

Input:
source = Pointer to input string (NULL terminated).
filler = Byte leading space filler.
len = Length of the right justification buffer.
Return:
None.

4.8.4 void memxor(BYTE * dest, BYTE * source, DWORD len)

Function:
To xor a memory buffer

Input:

source	= source pointer
dest	= destination pointer
len	= Length of data to be xor

Return:
None.

4.8.5 void rand_no(BYTE * dest)

Function:
To generate 8-byte random number

Input:
dest = destination pointer where the random to be saved

Return:
None.