Function reference

This document is intended to serve as a very short reference to useful `libepc` functions.

Screen handling

`void ClearScreen(BYTE8 attb)`
Erase screen setting each character to space and each character attribute to `'attb'`. A value of `0x07` specifies white characters on black background.

`void SetCursorPosition(int row, int col)`
Decide where the next character will be placed on screen.

`void SetCursorVisible(BOOL visible)`
Show or hide the text-cursor indicating current character position.

String handling

`void PutChar(char c)`
Display the single character `'c'`.

`void PutString(char* string)`
Display the *NUL-terminated* sequence of characters indicated by `'string'`.

`void PutUnsigned(unsigned n, int base, int width)`
Display an unsigned 32-bit integer in the given `'base'` using `'width'` characters.

`char* Unsigned2Ascii(char *bfr, unsigned val, int base)`
The `'val'` parameter is assumed to be written in `'base'` and it's string representation is stored in `'bfr'`. It is also assumed that `'bfr'` is large enough.

Timer functions

`QWORD64 CPU_Clock_Cycles(void)`
Number of clock cycles passed since last boot. Note that it return a *64-bit* value.

`DWORD32 Milliseconds(void)`
Number of milliseconds since program start.

`DWORD32 Now_Plus(int seconds)`
Number of milliseconds since program start plus `'seconds'`.

Window functions

```
void WindowCreate(char *title, int row_first, int row_last,
                  int col_first, int col_last)
```

Create an ASCII window on the screen with specified ‘title’. The remaining parameters specify where the four borders of the window shall be drawn.

```
void WindowSetCursor(WINDOW *w, int row, int col)
```

Place the text-cursor relative to the top-left corner of window ‘w’.

```
void WindowPutString(WINDOW *w, char *str)
```

Display the string ‘str’ starting at the current cursor position of window ‘w’.

```
void WindowPutChar(WINDOW *w, char ch)
```

Display the character ‘ch’ at the current cursor position of window ‘w’.

Queue functions

```
BOOL QueueCreate(int numb_items, int item_size)
```

Create a FIFO queue with capacity for ‘numb_items’. Each item is assumed to require ‘item_size’ bytes of storage in the queue.

```
BOOL QueueInsert(QUEUE *q, void *data)
```

Insert the ‘data’ item to the queue ‘q’. The size of ‘data’ must correspond to the size specified for ‘item_size’ when the queue was created.

```
BOOL QueueRemove(QUEUE *q, void *data)
```

Remove one item from the queue ‘q’ and place it in ‘data’. The allocated size of ‘data’ must correspond to the ‘item_size’ specified on queue creation.