PDT 3200

Run-Time Library
Reference
Manual

70-31577-01
Revision A — July, 1997

Symbol Technologies, Inc. One Symbol Plaza, Holtsville N.Y. 11742

PDT 3200
Run-Time Library Reference Manual

70-31577-01
Revision A
July, 1997

Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
http://www.symbol.com

Contents

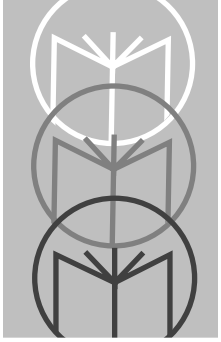
About This Document

Chapter 1. Library Cross-Reference

Chapter 2. Function Reference

Chapter 3. Serial Communications

Chapter 4. A Program Sample



About This Document

Overview

The Developer's Toolkit for the PDT 3200 DOS portable includes a run-time library. The functions in this library allow a developer to create C (or other high-level language) applications that take advantage of the special features of the PDT 3200. The functions contained in the library provide an API (application programming interface) to the BIOS services, serial communications capability, and bar code configuration driver.

Audience

This manual provides technical information that allows experienced C programmers to build full-featured applications for the PDT 3200.

Chapter Descriptions

Following is a list of the chapters in this document along with brief descriptions of the contents in each:

- *Chapter 1* describes how to install the run-time library files, and provides an overview of routines, header files, typedefs and constants included in the run-time library.
- *Chapter 2* provides function references.
- *Chapter 3* describes the serial communications function calls.
- *Chapter 4* provides a program example for Borland and Microsoft compilation.

Notational Conventions

The following conventions are used in this document:

- "You" refers to the programmer creating applications for the PDT 3200.
- Keystrokes in **bold** type within angle brackets indicate non-alphanumeric keystrokes on the PC or on the terminal. For example:

  Select the **<F1>** key on the terminal to access on-line help.
- **Bold** type is used:
  - for the names of services, functions, and commands
  - to identify menu items and input or text fields on a terminal screen
  - display command lines

  Note: Program command names and required parameters appear **without** brackets; optional parameters appear in square [] brackets.
- *Italics* are used:
  - for the names of parameters in function prototypes and variable names in usage and syntax descriptions
  - to highlight specific items in the general text
  - to identify chapters and sections in this and related documents
- Square brackets [] in a command line enclose optional command-line parameters.
- The piping symbol | represents "or" when used to separate command-line parameters on a command line; i.e., it separates alternative values for parameters.

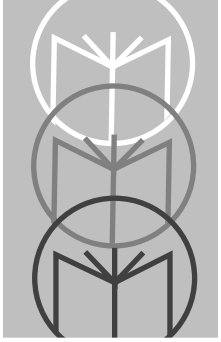
- Bullets (•) indicate:
  - action items
  - lists of alternatives
  - lists of required steps that are not necessarily sequential
- Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.

Related Publications

The following is a list of documents which provide more information on the PDT 3200 DOS Portable itself or the tools and utilities available for writing applications for the terminal.

Documents available from Symbol Technologies:

- *PDT 3200 Product Reference Guide,* p/n 70-31466-xx
- *PDT 3200 Technical Reference Guide,* p/n 70-31468-xx
- *PDT 3200 Quick Reference Guide*, p/n 70-31467-xx
- *PDT 3200 Run Time Library Reference Manual*, p/n 70-31577-xx
- *PDT 3200 Cradle Quick Reference Guide*, p/n 70-31469-xx



Chapter 1
Library Cross-Reference

Chapter Contents

Installing

To install the run-time library files, place the Developer's Toolkit disk in drive A, and run **`A:\SETUP`**.

Library Cross-Reference

This section provides an overview of the routines, header files, typedefs, and constants included in the run-time library. It also outlines the compilation requirements for programs that will include this library.

Header Files

Header File	Description
`filexfer.h`	Defines constants for **`SIO_FileTransfer()`**.
`p_bios.h`	Declares BIOS interface functions and corresponding constants.
`p_cfg.h`	Declares configuration programming functions and corresponding constants.
`p_supp.h`	Defines type declarations, constant definitions, register data structures, and compatibility macros, and declares internal function for API.
`sio.h`	Declares serial communications functions and corresponding constants.

Library

All of the routines in the run-time library are included in **`FALCS.LIB`**, **`FALCM.LIB`**, and **`FALCL.LIB`**. These libraries were created with the Borland C++ compiler, version 4.52. Modules in these libraries were compiled using the small-, medium-, and large-memory models, respectively. These libraries can be linked into applications using a Borland C++ or Microsoft C++ compiler. You must compile and link the applications with the correct memory model. If your application is compiled using the small-memory model, use **`FALCS.LIB`**. If your application is compiled using the medium-memory model, use **`FALCM.LIB`**. If your application is compiled using the large-memory model, use **`FALCL.LIB`**. See *Chapter 4: A Program Sample*.

TypeDefs

Variable Type	Equivalent C Type
BOOL	unsigned char
BYTE	unsigned char
DWORD	unsigned long
FLAGS	unsigned int
LPBYTE	unsigned char far*
LPWORD	unsigned long far*
PBYTE	unsigned char near*
PWORD	unsigned long near*
ticks_t	unsigned long
UCHAR	unsigned char
UINT	unsigned int
ULONG	unsigned long
USHORT	unsigned short
WORD	unsigned short

The following structure is used by the serial transfer routines. Constants for setting its members are provided in *Constant Definitions*. See *Chapter 3: Serial Communications* for detailed information about using the structure and its individual elements.

```
typedef struct {
                WORD  baseaddr;
                BYTE  portid;
                BYTE  databits;
                BYTE  stopbits;
                BYTE  parity;
                BYTE  breakbit;
                long  baudrate;
                BYTE  DTR;
                BYTE  RTS;
                BYTE  GPO1;
                BYTE  GPO2;
```

```
        BYTE  CTS;
        BYTE  DSR;
        BYTE  RI;
        BYTE  DCD;
        BYTE  protocol;
        WORD  commstatus;
        BYTE  interrupts;
        void  (ShowRx)(void);
        void  (ShowTx)(void);
} PORT;
```

Constant Definitions

Boolean Values

FALSE	0
TRUE	1

Video Service Constants

Video Modes

The following constants indicate video modes for **BIOS_SetVideoMode()**:

P_VMODE_GRAPHIC_320X200	0x04
P_VMODE_GRAPHIC_320X200_CB	0x05
P_VMODE_GRAPHIC_640X200	0x06
P_VMODE_TEXT_40X25	0x00
P_VMODE_TEXT_40X25_CB	0x01
P_VMODE_TEXT_80X25	0x02
P_VMODE_TEXT_80X25_CB	0x03
P_VMODE_TEXT_80X25_MONO	0x07

Cursor Style

The following constants indicate cursor styles for **BIOS_SetCursorStyle()**:

P_CUR_FULL	0x0007
P_CUR_HALF	0x0307
P_CUR_HTOP	0x0003
P_CUR_NORM	0x0607
P_CUR_OFF	0x0100

Screen Attribute

The following constants indicate the screen attribute for those functions receiving or returning the screen attribute:

P_ATT_INVERSE	0x70
P_ATT_NORM	0x07

Active Page Indicator

The following constant specifies the active video page for those functions that require the video page to be specified. If **P_ACTIVE_PAGE** is used instead of specifying the specific video page, the active page is determined by calling **BIOS_GetDisplayPage()**.

P_ACTIVE_PAGE	0xFF

Viewport Mode Setting

The following constants indicate the mode of viewport panning for **BIOS_GetViewportMode()** and **BIOS_SetViewportMode()**. If the value sent to **BIOS_SetViewportMode()** or returned by **BIOS_GetViewportMode()** equals **P_VIEWPT_HPAN + P_VIEWPT_VPAN (0x03)**, both horizontal and vertical panning are enabled.

P_VIEWPT_HPAN	0x01
P_VIEWPT_VPAN	0x02

Font Identifiers

The following constants check the value returned by **BIOS_GetFontInfo()**:

P_ACT_FONT	0x01
P_6X8_FONT	0x08
P_8X8_FONT	0x03
P_8X16_FONT	0x06

Standard PC Keyboard Flags

The following constants indicate the status of the **CTRL**, **ALT**, **CAPS LOCK**, and **INSERT** keys for **`BIOS_GetKeyFlags()`** and **`BIOS_GetKeyExtFlags()`**:

`P_KBDSHIFT_ALT`	`0x08`
`P_KBDSHIFT_CAPS`	`0x40`
`P_KBDSHIFT_CTRL`	`0x04`
`P_KBDSHIFT_INS`	`0x80`

Extended PC Keyboard Flags

The following constants indicate the status of **CTRL**, **ALT**, and **CAPS LOCK** keys for **`BIOS_GetKeyExtFlags()`**:

`P_KBDSHIFT_ALT_DOWN`	`0x0200`
`P_KBDSHIFT_CAPS_DOWN`	`0x4000`
`P_KBDSHIFT_CTRL_DOWN`	`0x0100`

Unit-Specific Keyboard Flags

The following constants indicate the status of "special" keys for **`BIOS_GetKeyInputState()`** or **`BIOS_SetKeyInputState()`**. Each constant represents a bit in the variable indicating key status. Therefore, multiple constants may be used simultaneously.

`P_KBDSTATE_ALPHA`	`0x0020`
`P_KBDSTATE_ALPHA_DOWN`	`0x2000`
`P_KBDSTATE_ALT`	`0x0008`
`P_KBDSTATE_ALT_DOWN`	`0x0800`
`P_KBDSTATE_CAPS`	`0x0010`
`P_KBDSTATE_CAPS_DOWN`	`0x1000`
`P_KBDSTATE_CTRL`	`0x0004`
`P_KBDSTATE_CTRL_DOWN`	`0x0400`
`P_KBDSTATE_FUNC1`	`0x0001`
`P_KBDSTATE_FUNC1_DOWN`	`0x0100`
`P_KBDSTATE_FUNC2`	`0x0002`
`P_KBDSTATE_FUNC2_DOWN`	`0x0200`
`P_KBDSTATE_INTL`	`0x0080`
`P_KBDSTATE_PRGM`	`0x0040`

Keyboard Click Settings

P_KBDSOUND_BEEP	0x02
P_KBDSOUND_CLICK	0x01
P_KBDSOUND_OFF	0x00

Special Key-Sequence Controls

P_KBDCTRL_PRGM	0x01
P_KBDCTRL_REBOOT	0x02

Speaker Services Constants

Volume Controls

P_SNDVOL_DOWN	0x09
P_SNDVOL_OFF	0x00
P_SNDVOL_UP	0x08

Advanced Power Management Constants

APM Return or Error Codes

The following constants check return values produced by the **BIOS_Apm...** library routines:

APM_ERR_CONNECTED	0x02
APM_ERR_DEVICEID	0x09
APM_ERR_DISABLED	0x01
APM_ERR_DISENGAGED	0x0B
APM_ERR_LOWBAT	0x81
APM_ERR_NOCONNECT	0x03
APM_ERR_NONE	0x00
APM_ERR_OUTOFRANGE	0x0A
APM_ERR_PENDING	0x80
APM_ERR_STATE	0x60
APM_ERR_UNKNOWN	0xFF

APM High-Speed Clock Speeds

The following constants are used when calling **BIOS_ApmHighSpeed()**:

APM_HSCLK_20MHZ	0x0002
APM_HSCLK_25MHZ	0x0003
APM_HSCLK_OFF	0x0000

APM_HSCLK_ON	0x0001

APM Low-Speed Clock Speeds

The following constants are used when calling **BIOS_ApmHighSpeed()**:

APM_CLK_1MHZ	0x0003
APM_CLK_2MHZ	0x0002
APM_CLK_4MHZ	0x0001
APM_CLK_9MHZ	0x0000

Communications Port Constants

Port Identifier

The following constants set port ID in the **PORT** data structure. Note that COM3 and COM4 are not yet supported:

SIO_COM1	0
SIO_COM2	1
SIO_MAX_PORTS	2

Format Settings

The following constants set parity in the **PORT** data structure:

SIO_EVEN_PARITY	0x18
SIO_MARK_PARITY	0x28
SIO_NO_PARITY	0x00
SIO_ODD_PARITY	0x08
SIO_SPACE_PARITY	0x38

The following constants set stop bits in the **PORT** data structure:

SIO_ONE_STOPBIT	0x00
SIO_ONE5_STOPBITS	0x00
SIO_TWO_STOPBITS	0x04

The following constants set data bits in the **PORT** data structure:

SIO_EIGHT_DATABITS	0x03
SIO_FIVE_DATABITS	0x00
SIO_SEVEN_DATABITS	0x02
SIO_SIX_DATABITS	0x01

Baud Rates

The following constants set baud rate in the **PORT** data structure:

SIO_BAUD_300	300
SIO_BAUD_1200	1200
SIO_BAUD_2400	2400
SIO_BAUD_4800	4800
SIO_BAUD_9600	9600
SIO_BAUD_19K	19200
SIO_BAUD_38K	38400L
SIO_BAUD_56K	57600L
SIO_BAUD_115K	115200L

Transfer Protocol

The following constants set protocol in the **PORT** data structure. The protocol setting is used by **SIO_FileTransfer()**.

SIO_ASCIIDUMP	0x03
SIO_PACKNAK	0x01
SIO_XMODEM	0x02

Interrupt Enabling

The following constants set interrupts in the **PORT** data structure:

SIO_INTR_NONE	0x00
SIO_INTR_READ	0x01

Error Codes

The following constants check the error code returned by **SIO_GetCommError()**:

SIO_BREAK_DETECT	0x010
SIO_DATAREADY	0x001
SIO_FRAMING_ERR	0x008
SIO_LINEERRORS	0x01e
SIO_OVERRUN_ERR	0x002
SIO_PARITY_ERR	0x004
SIO_READBUFFERFULL	0x100
SIO_TRANSBUFFEMPTY	0x020
SIO_TRANSEMPTY	0x040
SIO_WRITEBUFFERFULL	0x080

The following constants check the error code returned by **SIO_FileTransfer()**:

SIO_ERR_FILEAPPEND	-103
SIO_ERR_FILECREATE	-102
SIO_ERR_FILEOPEN	-101
SIO_ERR_FILEWRITE	-104
SIO_ERR_INVALIDFTP	-1
SIO_ERR_KEYSTROKE	-4
SIO_ERR_LINELENGTH	-201
SIO_ERR_MALLOC	-202
SIO_ERR_NONE	0
SIO_ERR_NOTDONE	-901
SIO_ERR_PACKETNBR	-204
SIO_ERR_TIMEOUT	-3
SIO_ERR_UNKNOWN	-900

File Transfer Parameters

The following constants set the **Flags** parameter in **SIO_FileTransfer()**:

SIO_FLG_APPEND	0x0004
SIO_FLG_RECEIVE	0x0002
SIO_FLG_TRANSMIT	0x0001

Flush Comm Buffer Parameters

The following constants set the **Flags** parameter in **SIO_FlushComm()**:

SIO_FLUSHALL	0x03
SIO_FLUSHREAD	0x01
SIO_FLUSHWRITE	0x02

Routines by Category

Video

BIOS_ClearScreen	p_bios.h
BIOS_ClearViewport	p_bios.h
BIOS_ClearWindow	p_bios.h
BIOS_CursorOff	p_bios.h
BIOS_CursorOn	p_bios.h
BIOS_DisableCursorEmulation	p_bios.h
BIOS_DisableVideo	p_bios.h

BIOS_EnableCursorEmulation	p_bios.h
BIOS_EnableVideo	p_bios.h
BIOS_GetCursorPos	p_bios.h
BIOS_GetCursorType	p_bios.h
BIOS_GetCurXY	p_bios.h
BIOS_GetDisplayPage	p_bios.h
BIOS_GetFontInfo	p_bios.h
BIOS_GetVideoColumns	p_bios.h
BIOS_GetVideoMode	p_bios.h
BIOS_GetViewportMode	p_bios.h
BIOS_GetViewportPos	p_bios.h
BIOS_GetViewportSize	p_bios.h
BIOS_IsCursorOn	p_bios.h
BIOS_Load6x8Font	p_bios.h
BIOS_Load8x8Font	p_bios.h
BIOS_Load8x16Font	p_bios.h
BIOS_LoadUserFont	p_bios.h
BIOS_MoveViewportDown	p_bios.h
BIOS_MoveViewportLeft	p_bios.h
BIOS_MoveViewportRight	p_bios.h
BIOS_MoveViewportUp	p_bios.h
BIOS_ReadPixel	p_bios.h
BIOS_RefreshDisplay	p_bios.h
BIOS_ScrollPageDown	p_bios.h
BIOS_ScrollPageUp	p_bios.h
BIOS_ScrollViewportDown	p_bios.h
BIOS_ScrollViewportUp	p_bios.h
BIOS_SetCursorPos	p_bios.h
BIOS_SetCursorStyle	p_bios.h
BIOS_SetCursorType	p_bios.h
BIOS_SetCurXY	p_bios.h
BIOS_SetDisplayPage	p_bios.h
BIOS_SetVideoMode	p_bios.h
BIOS_SetViewportMode	p_bios.h
BIOS_SetViewportPos	p_bios.h
BIOS_VideoReadCharAtt	p_bios.h
BIOS_VideoWriteCharAtt	p_bios.h
BIOS_VideoWriteCharOnly	p_bios.h
BIOS_WriteAttribStr	p_bios.h
BIOS_WriteAttribStrWithCur	p_bios.h

BIOS_WriteCharTeleMode	p_bios.h
BIOS_WritePixel	p_bios.h
BIOS_WriteString	p_bios.h
BIOS_WriteStringWithCur	p_bios.h
BIOS_WriteStrXY	p_bios.h

Advanced Power Management

BIOS_ApmAudioBLI	p_bios.h
BIOS_ApmBacklight	p_bios.h
BIOS_ApmGetStatus	p_bios.h
BIOS_ApmHighSpeed	p_bios.h
BIOS_ApmInitialize	p_bios.h
BIOS_ApmLowSpeed	p_bios.h
BIOS_ApmSleepDisable	p_bios.h
BIOS_ApmSleepEnable	p_bios.h
BIOS_ApmTimeout	p_bios.h
BIOS_ApmTimerInfo	p_bios.h

Serial Communications

SIO_AutoClose	sio.h
SIO_Close	sio.h
SIO_FileTransfer	sio.h
SIO_FlushComm	sio.h
SIO_GetComm	sio.h
SIO_GetCommError	sio.h
SIO_Keyflush	sio.h
SIO_Keyhit	sio.h
SIO_Open	sio.h
SIO_Read	sio.h
SIO_SetComm	sio.h
SIO_Ticks	sio.h
SIO_Time	sio.h
SIO_Write	sio.h

Keyboard

BIOS_GetKbdCtrl	p_bios.h
BIOS_GetKeyClick	p_bios.h
BIOS_GetKeyExtFlags	p_bios.h
BIOS_GetKeyFlags	p_bios.h

BIOS_GetKeyInputState	p_bios.h
BIOS_GetKeyStatus	p_bios.h
BIOS_KeyboardReadChar	p_bios.h
BIOS_KeyboardWriteChar	p_bios.h
BIOS_SetKbdCtrl	p_bios.h
BIOS_SetKeyClick	p_bios.h
BIOS_SetKeyInputState	p_bios.h

Time and Date

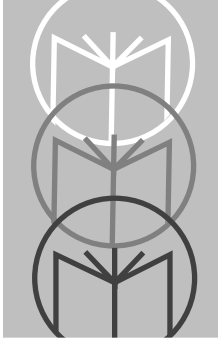
BIOS_GetDate	p_bios.h
BIOS_GetTickCount	p_bios.h
BIOS_GetTime	p_bios.h
BIOS_SetDate	p_bios.h
BIOS_SetTickCount	p_bios.h
BIOS_SetTime	p_bios.h

Sound

BIOS_SpeakerAlternate	p_bios.h
BIOS_SpeakerTone	p_bios.h
BIOS_GetSpeakerAlternate	p_bios.h
BIOS_GetSpeakerVolume	p_bios.h
BIOS_SetSpeakerAlternate	p_bios.h
BIOS_SetSpeakerVolume	p_bios.h

Configuration Programming

CFG_Read	p_cfg.h
CFG_Write	p_cfg.h



Chapter 2
Function Reference

Chapter Contents

BIOS_ApmAudioBLI

p_bios.h

Syntax

```
UCHAR BIOS_ApmAudioBLI(UCHAR Timeout);
```

Description

Sets the time, in minutes, between audio battery-low indications. Setting this timeout causes the unit to sound a tone every **Timeout** minutes if the battery is low. The audio warning uses extra current, causing the batteries to drain faster. Therefore, it is recommended that the warning be sounded as infrequently as possible (**Timeout** > 5) or that it not be used at all.

The audio battery low indicator does not sound until the battery has been in the low condition for the specified **Timeout** period.

Parameters

Timeout The time, in minutes, between audio signals that should be sounded if the battery is in a low condition. Valid values are from 0 to 255 minutes, with 0 (zero) disabling the audio battery low indicator. The indicator is disabled by default.

Returns

APM_ERR_NONE	Timeout was set successfully.
APM_ERR_UNKNOWN	Timeout could not be set.

Sample

```
#include <stdio.h>
#include “p_bios.h”

void Apm_Setup (BOOL FactorySettings)
{
    UINT   Auto_Off;
    UCHAR  Backlight;
    UCHAR  Audio_BLI;

    if (!FactorySettings)
    {
        // Power management timer settings.
        BIOS_ApmAudioBLI (20);        // Beep every 20 minutes when the
                                      // battery is low
        BIOS_ApmBacklight (10);       // Turn the backlight off after 10
                                      // seconds
        BIOS_ApmTimeout (240);        // Turn the unit off after 4
                                      // minutes of inactivity

        // Power management clock settings.
        BIOS_ApmHighSpeed(APM_HSCLK_OFF);     // Don’t allow high-speed
                                              // mode
        BIOS_ApmLowSpeed(APM_CLK_2MHZ);       // Low-speed clock to
                                              // 2.304 MHz
    }
    else
    {
        // Return to factory defaults.
        BIOS_ApmInitialize();
    }

    // Display the new timer settings
    BIOS_ApmTimerInfo(&Auto_Off, &Backlight, &Audio_BLI);
    BIOS_ClearScreen(P_ATT_NORM);
    printf (“Auto-Off  : %u sec \n”, Auto_Off);
    printf (“Audio Freq: %u min \n”, Audio_BLI);
    printf (“Backlight : %u sec \n", Backlight);
    return;
}
```

BIOS_ApmBacklight

p_bios.h

Syntax

```
UCHAR BIOS_ApmBacklight(UCHAR Timeout);
```

Description

Sets the auto timeout value, in seconds, for the backlight. After **Timeout** seconds with no keyboard activity, the backlight turns off. In this mode, the backlight is only suspended, not turned off; any keystroke turns the backlight back on. Trigger presses do not cause the backlight to come back on.

Parameters

Timeout The time, in seconds, that the backlight should remain on with no keyboard activity. Valid range is 0 to 255 seconds. 0 (zero) disables the timer, and the backlight will not timeout. The default timeout period is 15 seconds.

Returns

APM_ERR_NONE	Timeout was set successfully.
APM_ERR_UNKNOWN	Timeout could not be set.

Sample

See **BIOS_ApmAudioBLI**.

BIOS_ApmGetStatus

p_bios.h

Syntax

UCHAR BIOS_ApmGetStatus(void);

Description

Returns battery condition. A low battery condition indicates that less than 80% of battery power is remaining.

Parameters

None.

Returns

APM_ERR_NONE	Battery is not low.
APM_ERR_LOWBAT	Battery is low.
APM_ERR_UNKNOWN	Battery condition could not be checked.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void DisplayBatteryLow (void)
{
    UCHAR scanCode;
    UCHAR ch;

    if (BIOS_ApmGetStatus == APM_ERR_LOWBAT)
    {
        BIOS_ClearScreen(P_ATT_NORM);
        printf ("IMPORTANT: \nBattery Low\nChange Battery\n");
        printf (" < Hit any key >  ");
        while (!BIOS_GetKeyStatus(&scanCode, &ch));
        BIOS_KeyboardReadChar(&scanCode);
    }
    return;
}
```

BIOS_ApmHighSpeed

p_bios.h

Syntax

```
UCHAR BIOS_ApmHighSpeed(UCHAR ClockSpeed);
```

Description

Enables or disables the ability to enter high-speed mode and sets the clock frequency of high-speed mode. This routine does not put the unit in high-speed mode. This capability is useful for the developer who knows that the application will not be CPU intensive and wishes to save power by preventing transitions to high-speed mode. If high-speed mode is disabled, the high-speed PLL is turned off, and the high-speed clock is run from the low-speed PLL at 9.216 MHz. Setting the clock speed for high-speed mode will automatically enable high-speed mode.

Parameters

ClockSpeed Indicates whether to enable or disable high-speed mode. Alternatively this parameter can indicate the speed of the clock in high-speed mode.
APM_HSCLK_OFF disables high-speed mode.
APM_HSCLK_ON enables high-speed mode.
APM_HSCLK_20MHZ sets clock speed to 20 MHz in high-speed mode and enables high-speed mode.
APM_HSCLK_25MHZ sets clock speed to 25 MHz in high-speed mode and enables high-speed mode.

Returns

APM_ERR_NONE	No error.
APM_ERR_OUTOFRANGE	**ClockSpeed** parameter is out of range.
APM_ERR_UNKNOWN	Unknown error.

Sample

See **BIOS_ApmAudioBLI**.

BIOS_ApmInitialize

p_bios.h

Syntax

```
UCHAR BIOS_ApmInitialize(void);
```

Description

Restores the advanced power management system to its power-up defaults. Audio battery-low indicator is disabled, backlight timeout is set to 15 seconds, high-speed mode is enabled with clock speed set to 25MHz, and auto-off timeout is set to 5 minutes.

Parameters

None.

Returns

APM_ERR_NONE	No error.
APM_ERR_UNKNOWN	Unknown error.

Sample

See **BIOS_ApmAudioBLI**.

BIOS_ApmLowSpeed

p_bios.h

Syntax

```
UCHAR BIOS_ApmLowSpeed(UCHAR ClockSpeed);
```

Description

Sets the low-speed PLL clock frequency. Note that the CPU internal speed is half of the low-speed PLL. Only the low-speed PLL is affected, not the high-speed PLL.

Parameters

ClockSpeed Indicates the speed of the clock in low-speed mode.
APM_CLK_9MHZ sets clock speed to 9.216 MHz and internal CPU speed to 4.608 MHz.
APM_CLK_4MHZ sets clock speed to 4.608 MHz and internal CPU speed to 2.304 MHz.
APM_CLK_2MHZ sets clock speed to 2.304 MHz and internal CPU speed to 1.152 MHz.
APM_CLK_1MHZ sets clock speed to 1.152 MHz and internal CPU speed to 0.576 MHz.

Returns

APM_ERR_NONE	No error.
APM_ERR_OUTOFRANGE	**ClockSpeed** parameter is out of range.
APM_ERR_UNKNOWN	Unknown error.

Sample

See **BIOS_ApmAudioBLI**.

BIOS_ApmSleepDisable

p_bios.h

Syntax

```
UCHAR BIOS_ApmSleepDisable(void);
```

Description

Forces the CPU to remain on. Call this function before entering a CPU-intensive task. When the task is complete, **`BIOS_ApmSleepEnable()`** should be called to allow the unit to continue conserving power. Be very careful when using **`BIOS_ApmSleepDisable()`**; it can have a severe impact on battery life.

Parameters

None.

Returns

`APM_ERR_NONE`	CPU will remain on.
`APM_ERR_UNKNOWN`	Unknown error.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void SleepDisabled(void)
{
    UCHAR scanCode;
    UCHAR ch;

    // Disable sleep mode in anticipation of CPU
    // intensive task
    BIOS_ApmSleepDisable();
    BIOS_ClearScreen(P_ATT_NORM);
    printf ("Sleep Disabled\n");
    printf ("  < Hit any key >  ");
    while (!BIOS_GetKeyStatus(&scanCode, &ch));
    BIOS_KeyboardReadChar(&scanCode);

    // CPU intensive code here

    // IMPORTANT: Enable sleep mode so unit can
    // maximize power conservation.
```

```
    BIOS_ApmSleepEnable();
    BIOS_ClearScreen(P_ATT_NORM);
    printf ("\nSleep Enabled\n");
    printf ("  < Hit any key >  ");
    while (!BIOS_GetKeyStatus(&scanCode, &ch));
    BIOS_KeyboardReadChar(&scanCode);

    return;
}
```

BIOS_ApmSleepEnable

p_bios.h

Syntax

```
UCHAR BIOS_ApmSleepEnable(void);
```

Description

Enables the ability of the unit to shut off the CPU for power conservation. This is the default state of the unit. Do not call this function unless the capability has been disabled with **`BIOS_ApmSleepDisable()`**.

Parameters

None.

Returns

`APM_ERR_NONE`	CPU may now turn off.
`APM_ERR_UNKNOWN`	Unknown error.

Sample

See **`BIOS_ApmSleepDisable`**.

BIOS_ApmTimeout

p_bios.h

Syntax

```
UCHAR BIOS_ApmTimeout(UINT Timeout);
```

Description

Sets the auto timeout value, in seconds, for the unit. After **Timeout** seconds with no activity, the unit will turn off.

Parameters

Timeout The time, in seconds, that the unit should remain on with no activity. Valid range is 16 to 1032 seconds (17 minutes, 12 seconds). 0 (zero) disables the timer, and the unit does not timeout. 300 seconds (5 minutes) is the default timeout period. The resolution is to 4-second intervals, and so the value entered is rounded down to the nearest multiple of 4. For example, values of 17, 18, or 19 produce a 16-second timeout.

Returns

APM_ERR_NONE	No error.
APM_ERR_OUTOFRANGE	Timeout parameter is out of range.
APM_ERR_UNKNOWN	Unknown error.

Sample

See **BIOS_ApmAudioBLI**.

BIOS_ApmTimerInfo

p_bios.h

Syntax

```
UCHAR BIOS_ApmTimerInfo(UINT *Auto_Off, UCHAR *Backlight, UCHAR
*Audio_BLI);
```

Description

Returns power management timer settings.

Parameters

None.

Returns

Auto_Off	Auto-off timeout in seconds.
Backlight	Backlight auto-off timeout in seconds.
Audio_BLI	Audio battery-low indicator frequency in minutes.
APM_ERR_NONE	No error.
APM_ERR_UNKNOWN	Unknown error.

Sample

See **BIOS_ApmAudioBLI**.

BIOS_ClearScreen

p_bios.h

Syntax

```
void BIOS_ClearScreen(UINT Attrib);
```

Description

Clears the logical display screen, filling the cleared area with the specified attribute. Repositions the cursor to the top of the screen.

This is the equivalent of executing **CLS** from a DOS prompt. The screen width used by this routine is the same as that returned by **BIOS_GetVideoColumns()**. The screen length used is the same as that returned in the **LastRow** parameter of **BIOS_GetFontInfo()**.

Parameters

Attrib Screen attribute that is to be written to the screen as the screen is being cleared. **P_ATT_NORM** specifies a normal white-on-black display attribute. **P_ATT_INVERSE** specifies a reverse-video (black-on-white) display attribute.

Returns

None.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void ClearDisplayFunctions (void)
{
    char Key;

    // Enable horizontal and vertical panning
    // so we can look at what is cleared
    BIOS_SetViewportMode (P_VIEWPT_HPAN | P_VIEWPT_VPAN);

    // Fill and perform BIOS_ClearScreen
    // Clears entire logical screen
    FillScreen();
    BIOS_ClearScreen(P_ATT_NORM);
```

```
    scanf ("%c", &Key);

    // Fill and perform BIOS_ClearViewport
    // Clears only viewport. Panning down or right
    // will reveal text outside of viewport
    FillScreen();
    BIOS_ClearViewport(P_ATT_NORM);
    scanf ("%c", &Key);

    // Fill and perform BIOS_ClearWindow
    // Clear a square in the middle of the screen
    // viewport
    FillScreen();
    BIOS_ClearWindow(P_ATT_NORM, 2, 5, 5, 15);
    scanf ("%c", &Key);
}


void FillScreen (void)
{
    UINT BotRow, RgtCol;
    UINT CharWidth;
    UINT CharHeight;
    UCHAR far *FontTable;

    int i, j;
    char str[81];
    char Key;

    // Use GetFontInfo and GetVideoColumns to determine screen size.
    BIOS_ClearScreen(P_ATT_NORM);
    BIOS_GetFontInfo(P_ACT_FONT, &CharWidth, &CharHeight,
        &BotRow, &FontTable);
    RgtCol = BIOS_GetVideoColumns();

    // Fill screen with varying text on each line
    for (i=0; i<=BotRow; i++)
    {
        for (j=0; j<RgtCol-1; j++)
            str[j] = 65+i+j;
        str[RgtCol-1] = '\0';
        BIOS_WriteStrXY (i, 0, str);
        if (i < BotRow) printf ("\n");
    }

    // Reset cursor to top left
    BIOS_SetCurXY(0,0);
    scanf ("%c", &Key);
}
```

BIOS_ClearViewport

p_bios.h

Syntax

```
void BIOS_ClearViewport(UINT Attrib);
```

Description

Clears the physical viewport, filling the cleared area with the given screen attribute.

The viewport size used by this routine is the same as that returned by **BIOS_GetViewportSize()**.

Parameters

Attrib Screen attribute that is to be written as the viewport is being cleared. **P_ATT_NORM** specifies a normal white-on-black display attribute. **P_ATT_INVERSE** specifies a reverse-video (black-on-white) display attribute.

Returns

None.

Sample

See **BIOS_ClearScreen()**.

BIOS_ClearWindow

p_bios.h

Syntax

```
void BIOS_ClearWindow(UINT Attrib, UINT TRow, UINT LCol, UINT
BRow, UINT RCol);
```

Description

Clears the window specified, filling the cleared area with the given screen attribute.

Parameters

Attrib	Screen attribute that is to be written as the viewport is being cleared. **P_ATT_NORM** specifies a normal white-on-black display attribute. **P_ATT_INVERSE** specifies a reverse-video (black-on-white) display attribute.
TRow	Upper row of the region to be cleared.
LCol	Left column of the region to be cleared.
BRow	Lower row of the region to be cleared.
RCol	Right column of the region to be cleared.

Note: The row and column numberings both begin at 0.

Returns

None.

Sample

See **BIOS_ClearScreen()**.

BIOS_CursorOff

p_bios.h

Syntax

```
void BIOS_CursorOff(void);
```

Description

Turns off the cursor on the video display. The current cursor style is maintained while the cursor is not visible. Calling **`BIOS_CursorOn()`** restores the cursor.

Parameters

None.

Returns

None.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void Test_BIOS_Cursor (void)
{
    UINT old_start, old_end;
    UINT test;
    char Key;

    BIOS_ClearScreen(P_ATT_NORM);
    printf ("Sample Code\n");
    printf ("for Cursor\n");
    printf ("control.\n");

    // Preserve starting state
    BIOS_GetCursorType(&old_start,&old_end);

    // Full size
    BIOS_SetCursorStyle(P_CUR_FULL);
    printf ("\nP_CUR_FULL...");
    scanf("%c",&Key);

    // Half size
    BIOS_SetCursorStyle(P_CUR_HALF);
    printf ("\nP_CUR_HALF...");
```

```
    scanf("%c",&Key);

    // Turn it off
    BIOS_CursorOff();
    printf ("\nOFF...");
    scanf("%c",&Key);

    // Check that it's off
    test = BIOS_IsCursorOn();
    printf ("\nState:%u  1=On 0=Off",test);
    scanf("%c",&Key);

    // Turn it back on
    BIOS_CursorOn();
    printf ("\nP_CUR_HALF...");
    scanf("%c",&Key);

    // Check that it's on
    test = BIOS_IsCursorOn();
    printf ("\nState:%u  1=On 0=Off",test);
    scanf("%c",&Key);

    // Top half
    BIOS_SetCursorStyle(P_CUR_HTOP);
    printf ("\nP_CUR_HTOP...");
    scanf("%c",&Key);

    // Back to normal using scan lines
    BIOS_SetCursorStyle(P_CUR_NORM);
    printf ("\nP_CUR_NORM...");
    scanf("%c",&Key);

    // Restore to original
    BIOS_SetCursorType(old_start,old_end);
    printf ("\nOriginal");
    scanf("%c",&Key);
}
```

BIOS_CursorOn

p_bios.h

Syntax

```
void BIOS_CursorOn(void);
```

Description

Turns on the cursor using the current cursor style. The current cursor style is maintained while the cursor is off, so the last visible cursor is the style restored.

Parameters

None.

Returns

None.

Sample

See **`BIOS_CursorOff()`**.

BIOS_DisableCursorEmulation

p_bios.h

Syntax

`void BIOS_DisableCursorEmulation(void);`

Description

Disables mapping of cursor start and end lines. When cursor emulation is disabled, **`BIOS_SetCursorType()`** sets the exact size of the cursor. When cursor emulation is enabled, the start and end lines passed to **`BIOS_SetCursorType()`** are converted to result in predefined cursor sizes **`(P_CURR_NORM, P_CURR_HALF, P_CURR_HTOP, P_CURR_FULL)`**.

Parameters

None.

Returns

None.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void Test_BIOS_CursorEmulation (void)
{
    UINT old_start, old_end;
    UINT test;
    char Key;

    BIOS_ClearScreen(P_ATT_NORM);

    // Preserve starting state
    BIOS_GetCursorType(&old_start,&old_end);

    // Cursor Emulation results in full
    // cursor when StartLine is 2 and Endline is 7
    BIOS_EnableCursorEmulation();
    BIOS_SetCursorType(2,7);
    printf ("Emulation on...");
    scanf("%c",&Key);
```

```
    // Without emulation, the cursor
    // uses actual lines provided
    BIOS_DisableCursorEmulation();
    BIOS_SetCursorType(2,7);
    printf ("Emulation off...");
    scanf("%c",&Key);

    // Restore to original
    BIOS_SetCursorType(old_start,old_end);
}
```

BIOS_DisableVideo

p_bios.h

Syntax

```
void BIOS_DisableVideo(void);
```

Description

Turns off the video display. Disabling the video has no effect on the contents of video RAM. Information printed to the display while video is disabled is written to video RAM. It appears when the video is enabled using **`BIOS_EnableVideo()`**.

Parameters

None.

Returns

None.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void VideoOnOff (void)
{
    char Key;

    BIOS_ClearScreen(P_ATT_NORM);

    // Display first line of text
    printf ("Any key to turn off\n");
    printf ("Again to turn back on\n");
    scanf("%c",&Key);

    // Turn off the video display
    BIOS_DisableVideo();

    // Send text to video RAM
    printf ("\nEntered while off");
    scanf("%c",&Key);

    // Turn on the video display
    BIOS_EnableVideo();
    scanf("%c",&Key);
}
```

BIOS_EnableCursorEmulation

p_bios.h

Syntax

```
void BIOS_EnableCursorEmulation(void);
```

Description

Enables mapping of cursor start and end lines. When cursor emulation is disabled, **BIOS_SetCursorType()** sets the exact size of the cursor. When cursor emulation is enabled, the start and end lines passed to **BIOS_SetCursorType()** are converted to result in predefined cursor sizes **(P_CURR_NORM, P_CURR_HALF, P_CURR_HTOP, P_CURR_FULL)**.

Parameters

None.

Returns

None.

Sample

See **BIOS_DisableCursorEmulation()**.

BIOS_EnableVideo

p_bios.h

Syntax

```
void BIOS_EnableVideo(void);
```

Description

Turns on the video display. Enabling the video has no effect on the contents of video RAM. Information printed to the display while video is disabled is written to video RAM. It appears when the video is enabled using **`BIOS_EnableVideo()`**.

Parameters

None.

Returns

None.

Sample

See **`BIOS_DisableVideo()`**.

BIOS_GetCursorPos

p_bios.h

Syntax

```
void BIOS_GetCursorPos(UINT DisplayPage, UINT *Row, UINT
*Column);
```

Description

Returns the position of the cursor on a specified display page.

Parameters

DisplayPage Video display page of cursor to check.
P_ACTIVE_PAGE indicates the current active display page.

Returns

Row Row position of cursor.
Column Column position of cursor.

Note: The row and column numberings both begin at 0.

Sample

See **BIOS_GetDisplayPage()**.

BIOS_GetCursorType

p_bios.h

Syntax

```
void BIOS_GetCursorType(UINT *StartLine, UINT *EndLine);
```

Description

Returns the start and end lines of the current cursor.

Parameters

None.

Returns

`StartLine`	Indicates the top line of the cursor.
`EndLine`	Indicates the bottom line of the cursor.

Sample

See **`BIOS_CursorOff()`** and **`BIOS_DisableCursorEmulation()`**.

BIOS_GetCurXY

p_bios.h

Syntax

```
void BIOS_GetCurXY(UINT *Row, UINT *Column);
```

Description

Returns the position of the cursor on the active display page. This is a macro equivalent to **`BIOS_GetCursorPos(P_ACTIVE_PAGE, Row, Column)`**.

Parameters

None.

Returns

`Row`	Row position of cursor.
`Column`	Column position of cursor.

Note: The row and column numberings both begin at 0.

Sample

See **`BIOS_GetDisplayPage()`**.

BIOS_GetDate

p_bios.h

Syntax

```
void BIOS_GetDate(UINT *Century, UINT *Year, UINT *Month, UINT
*Day);
```

Description

Returns the current system date.

Parameters

None.

Returns

Century	Century (19 or 20).
Year	Year (0–99).
Month	Month (1–12).
Day	Day (1–31).

Sample

```
#include <stdio.h>
#include "p_bios.h"

void DateAndTime(void)
{
    UCHAR ScanCode;
    UCHAR Input;
    UINT  Century,Year,Month,Day;
    UINT  Hours,Minutes,Seconds,DST;

    // Get current system date and time
    BIOS_GetDate(&Century, &Year, &Month, &Day);
    BIOS_GetTime(&Hours, &Minutes, &Seconds, &DST);

    BIOS_ClearScreen(P_ATT_NORM);
    printf ("\n%2u:%2u:%2u DST:%u", Hours, Minutes, Seconds, DST);
    printf ("\n%u/%u/%u%u", Month, Day, Century, Year);
    printf ("\n\n Press any key...");

    BIOS_KeyboardReadChar(&ScanCode);

    // Change for daylight savings
    BIOS_GetTime(&Hours, &Minutes, &Seconds, &DST);
```

```
    BIOS_SetTime(Hours+1, Minutes, Seconds, DST);
    return;
}
```

BIOS_GetDisplayPage

p_bios.h

Syntax

```
UINT BIOS_GetDisplayPage(void);
```

Description

Returns the currently active display page.

Parameters

None.

Returns

Active display page in the range 0 to 7.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void GetActivePage(void)
{
    UINT Row, Column;
    UINT DisplayPage;

    // Two ways to get the cursor position
    // on the active page
    // Get the active page and call BIOS_GetCursorPos()
    DisplayPage = BIOS_GetDisplayPage();
    BIOS_GetCursorPos (DisplayPage, &Row, &Column);

    // Use the BIOS_GetCurXY macro. It gets the active page
    // for you.
    BIOS_GetCurXY (&Row, &Column);
    return;
}
```

BIOS_GetFontInfo

p_bios.h

Syntax

```
void BIOS_GetFontInfo(UINT FontId, UINT *CharWidth, UINT
*CharHeight, UINT *LastRow, UCHAR far **FontTable);
```

Description

Returns information about the font specified by **FontId**.

Parameters

FontId Font identifier.
P_ACT_FONT gets information for the currently active font.
P_8X8_FONT gets information for the 8x8 font.
P_8X16_FONT gets information for the 8x16 font.
P_6X8_FONT gets information for the 6x8 font.

Returns

CharWidth	Pixel width of display character.
CharHeight	Bytes per character.
LastRow	Last row of logical screen. The first row is 0.
FontTable	Address to font table containing character bitmaps.

Sample

See **BIOS_ClearScreen()**.

BIOS_GetKbdCtrl

p_bios.h

Syntax

```
UCHAR BIOS_GetKbdCtrl(void);
```

Description

Returns the status of the keyboard control flags. These flags enable or disable special key sequences.

Parameters

None.

Returns

Bit flags defined by the following constants. These flags may be combined in any logical combination.

P_KBDCTRL_PRGM	Trigger programming is enabled.
P_KBDCTRL_REBOOT	**CTRL-ALT-DEL** key sequence is enabled.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void SetMiscConfigs(void)
{
    int keyClick;
    UCHAR kbdCtrl;
    UINT altTone;
    UINT volume;
    UCHAR ScanCode;

    // Key click, trigger programming disabled, CTRL-ALT-DEL enabled.
    // Alternate speaker tone at lowest frequency.
    // Turn down the volume a bit (default is 7).
    BIOS_SetKeyClick (P_KBDSOUND_CLICK);
    BIOS_SetKbdCtrl (P_KBDCTRL_REBOOT);
    BIOS_SetSpeakerAlternate(0);
    BIOS_SetSpeakerVolume(4);

    // Use access functions to get the new settings.
```

```
    BIOS_ClearScreen(P_ATT_NORM);
    keyClick = BIOS_GetKeyClick();
    if (keyClick == P_KBDSOUND_CLICK)
        printf ("Keyboard will click\n");
    else if (keyClick == P_KBDSOUND_BEEP)
        printf ("Keyboard will beep\n");
    else
        printf ("Keyboard is silent\n");

    kbdCtrl = BIOS_GetKbdCtrl();
    if (kbdCtrl & P_KBDCTRL_PRGM)
        printf ("Trigger pgm On\n");
    else
        printf ("Trigger pgm Off\n");

    if (kbdCtrl & P_KBDCTRL_REBOOT)
        printf ("CTRL-ALT-DEL On\n");
    else
        printf ("CTRL-ALT-DEL Off\n");

    printf ("Alt Tone=%u\n", BIOS_GetSpeakerAlternate());
    printf ("Volume=%u\n", BIOS_GetSpeakerVolume());
    printf ("\nPress any key...");
    BIOS_KeyboardReadChar(&ScanCode);
    return;
}
```

BIOS_GetKeyClick

p_bios.h

Syntax

```
UCHAR BIOS_GetKeyClick(void);
```

Description

Returns the keyboard click setting.

Parameters

None.

Returns

P_KBDSOUND_OFF	Keypresses generate no sound.
P_KBDSOUND_CLICK	Keypresses generate mechanical click.
P_KBDSOUND_BEEP	Keypresses generate beep.

Sample

See **BIOS_GetKbdCtrl()**.

BIOS_GetKeyExtFlags

p_bios.h

Syntax

```
UINT BIOS_GetKeyExtFlags(void);
```

Description

Returns the status of **CTRL**, **ALT**, and **CAPS LOCK** keys. The only additional information this function provides over **`BIOS_GetKeyFlags()`** is whether a **CAPS LOCK**, **CTRL**, or **ALT** key is being held down. **`BIOS_GetKeyInputState()`**, which handles the PDT 3200 keyboard better, is recommended for these purposes.

This function is provided for PC compatibility. The **CTRL** and **ALT** keys on the unit are considered to be **LEFT-CTRL** and **LEFT-ALT** keys.

Parameters

None.

Returns

Bit flags defined by the following constants. These flags may be combined in any logical combination.

`P_KBDSHIFT_CTRL`	**CTRL** key is pressed.
`P_KBDSHIFT_ALT`	**ALT** key is pressed.
`P_KBDSHIFT_CAPS`	**CAPS LOCK** is active.
`P_KBDSHIFT_INS`	**INS** is active.
`P_KBDSHIFT_CTRL_DOWN`	**CTRL** key is pressed.
`P_KBDSHIFT_ALT_DOWN`	**ALT** key is pressed.
`P_KBDSHIFT_CAPS_DOWN`	**CAPS LOCK** key is pressed and active.

Sample

See **`BIOS_GetKeyFlags()`**

BIOS_GetKeyFlags

p_bios.h

Syntax

```
UCHAR BIOS_GetKeyFlags(void);
```

Description

Returns the status of **CTRL**, **ALT**, **CAPS LOCK**, and **INS** keys. **`BIOS_GetKeyInputState()`**, which handles the special keyboard of the unit better, is recommended for these purposes.

Parameters

None.

Returns

Bit flags defined by the following constants. These flags may be combined in any logical combination.

`P_KBDSHIFT_CTRL`	**CTRL** key is pressed.
`P_KBDSHIFT_ALT`	**ALT** key is pressed.
`P_KBDSHIFT_CAPS`	**CAPS LOCK** is active.
`P_KBDSHIFT_INS`	**INS** is active.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void KeyStatusFlags(void)
{
    int stateChanges = 0;
    UCHAR flags, oldflags=0;
    UINT extflags, oldextflags=0;

    BIOS_ClearScreen(P_ATT_NORM);

    // Display status of special keys through 50 state change
    while (stateChanges < 50)
    {
        flags = BIOS_GetKeyFlags();
```

```
        extflags = BIOS_GetKeyExtFlags();

        if ((flags != oldflags) || (extflags != oldextflags))
        {
            oldflags = flags;
            oldextflags = extflags;
            stateChanges++;
            BIOS_SetCurXY(0,0);
            printf ("Flags:\n");
            if (flags&P_KBDSHIFT_CTRL) printf ("CTRL ");
                else printf ("    ");
            if (flags&P_KBDSHIFT_ALT) printf ("ALT ");
                else printf ("    ");
            if (flags&P_KBDSHIFT_INS) printf ("INS ");
                else printf ("    ");
            if (flags&P_KBDSHIFT_CAPS) printf ("CAP ");
                else printf ("    ");
            printf ("\n");

            printf ("Ext Flags:\n");
            if (extflags&P_KBDSHIFT_CTRL) printf ("CTRL ");
                else printf ("    ");
            if (extflags&P_KBDSHIFT_ALT) printf ("ALT ");
                else printf ("    ");
            if (extflags&P_KBDSHIFT_INS) printf ("INS ");
                else printf ("    ");
            if (extflags&P_KBDSHIFT_CAPS) printf ("CAP ");
                else printf ("    ");
            printf ("\n");

            printf ("DOWN:\n");
            if (extflags&P_KBDSHIFT_CTRL_DOWN) printf ("CTRL ");
                else printf ("    ");
            if (extflags&P_KBDSHIFT_ALT_DOWN) printf ("ALT ");
                else printf ("    ");
            if (extflags&P_KBDSHIFT_CAPS_DOWN) printf ("CAPS ");
                else printf ("    ");
            printf ("\n");
        }
    }
    return;
}
```

BIOS_GetKeyInputState

p_bios.h

Syntax

```
UINT BIOS_GetKeyInputState(void);
```

Description

Checks the keyboard buffer to see if a keystroke is waiting to be read. Any keystroke waiting is read and returned. The keystroke is not removed from the keyboard buffer.

Parameters

None.

Returns

Bit flags defined by the following constants. These flags may be combined in any logical combination.

P_KBDSTATE_ALPHA_DOWN	**ALPHA** key is pressed.
P_KBDSTATE_CAPS_DOWN	**CAPS LOCK** key is pressed.
P_KBDSTATE_ALT_DOWN	**ALT** key is pressed.
P_KBDSTATE_CTRL_DOWN	**CTRL** key is pressed.
P_KBDSTATE_FUNC2_DOWN	**FN 2** key is pressed.
P_KBDSTATE_FUNC1_DOWN	**FN 1** key is pressed.
P_KBDSTATE_INTL	Unit is in international-character entry mode.
P_KBDSTATE_PRGM	Unit is in programming mode.
P_KBDSTATE_ALPHA	Unit is in alpha mode.
P_KBDSTATE_CAPS	**CAPS LOCK** is active.
P_KBDSTATE_ALT	**ALT** is active.
P_KBDSTATE_CTRL	**CTRL** is active.
P_KBDSTATE_FUNC2	**FN 2** is active.
P_KBDSTATE_FUNC1	**FN 1** is active.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void UnitKeyStatusFlags(void)
{
    int stateChanges = 0;
    UINT flags, oldflags = 0;

    BIOS_ClearScreen(P_ATT_NORM);

    // Display status of special keys through 50 state changes
    while (stateChanges < 50)
    {
        flags = BIOS_GetKeyInputState();
        if (flags != oldflags)
        {
            oldflags = flags;
            stateChanges++;
            BIOS_SetCurXY(0,0);
            if (flags&P_KBDSTATE_CTRL) printf ("CTRL ");
                else printf ("     ");
            if (flags&P_KBDSTATE_ALT) printf ("ALT ");
                else printf ("     ");
            if (flags&P_KBDSTATE_FUNC1) printf ("FN1 ");
                else printf ("     ");
            if (flags&P_KBDSTATE_FUNC2) printf ("FN2 ");
                else printf ("     ");
            printf ("\n");
            if (flags&P_KBDSTATE_CAPS) printf ("CAP ");
                else printf ("     ");
            if (flags&P_KBDSTATE_ALPHA) printf ("ALP ");
                else printf ("     ");
            if (flags&P_KBDSTATE_PRGM) printf ("PGM ");
                else printf ("     ");
            if (flags&P_KBDSTATE_INTL) printf ("INTL");
                else printf ("     ");
            printf ("\n");

            printf ("DOWN:\n");
            if (flags&P_KBDSTATE_CTRL_DOWN) printf ("CTRL ");
                else printf ("     ");
            if (flags&P_KBDSTATE_ALT_DOWN) printf ("ALT ");
                else printf ("     ");
            if (flags&P_KBDSTATE_FUNC1_DOWN) printf ("FN1 ");
                else printf ("     ");
            if (flags&P_KBDSTATE_FUNC2_DOWN) printf ("FN2 ");
                else printf ("     ");
            printf ("\n");
            if (flags&P_KBDSTATE_CAPS_DOWN) printf ("CAP ");
                else printf ("     ");
```

```
            if (flags&P_KBDSTATE_ALPHA_DOWN) printf ("ALP ");
                else printf ("    ");
            printf ("\n");
        }
    }
    // Clear all states
    BIOS_SetKeyInputState (0);
    return;
}
```

BIOS_GetKeyStatus

p_bios.h

Syntax

```
int BIOS_GetKeyStatus(UCHAR *ScanCode, UCHAR *Ch);
```

Description

Checks the keyboard buffer to see if a keystroke is waiting to be read. Any keystroke waiting is read and returned. The keystroke is not removed from the keyboard buffer.

Parameters

None.

Returns

ScanCode	Scan code of the key pressed.
Char	ASCII value of key pressed.
FALSE	Keystroke is not waiting.
TRUE	Keystroke is waiting.

Sample

See **BIOS_APMGetStatus**.

BIOS_GetSpeakerAlternate

p_bios.h

Syntax

```
UCHAR BIOS_GetSpeakerAlternate(void);
```

Description

Returns the alternate speaker tone setting.

Parameters

None.

Returns

0–7	Index into alternate speaker tone table. The table designates specific frequencies. 0 is the lowest frequency; 7 is the highest.

Sample

See **BIOS_GetKbdCtrl()**.

BIOS_GetSpeakerVolume

p_bios.h

Syntax

```
UCHAR BIOS_GetSpeakerVolume(void);
```

Description

Returns the speaker volume setting.

Parameters

None.

Returns

P_SNDVOL_OFF	Speaker is turned off.
1–7	Volume settings with 1 being the softest and 7 being the loudest.

Sample

See **BIOS_GetKbdCtrl()**.

BIOS_GetTickCount

p_bios.h

Syntax

```
ULONG BIOS_GetTickCount(UINT *PastMidnight);
```

Description

Returns the number of clock ticks since midnight. If midnight has been passed since last read, the **`PastMidnight`** flag is set.

Parameters

None.

Returns

`PastMidnight`	Flag indicating that midnight was passed since last read.
`Clock Ticks`	Clock ticks since midnight (0–1,573,040).

Sample

None.

BIOS_GetTime

p_bios.h

Syntax

```
void BIOS_GetTime(UINT *Hours, UINT *Minutes, UINT *Seconds,
UINT *DST);
```

Description

Returns the system time.

Parameters

None.

Returns

Hours	Hours (0–23).
Minutes	Minutes (0–59).
Seconds	Seconds (0–59).
DST	Daylight savings time flag. FALSE (0) if not daylight savings time; TRUE (1) if it is daylight savings time.

Sample

See **BIOS_GetDate()**.

BIOS_GetVideoColumns

p_bios.h

Syntax

```
UINT BIOS_GetVideoColumns(void);
```

Description

Returns the number of character columns available for the current video mode.

Parameters

None.

Returns

Columns	Number of columns available. This is typically 40 or 80.

Sample

See **BIOS_ClearScreen()**.

BIOS_GetVideoMode

p_bios.h

Syntax

```
UINT BIOS_GetVideoMode(void);
```

Description

Returns the current video mode. The default video mode is **P_VMODE_TEXT_80x25_CB**.

Some video modes are provided for compatibility only.

P_VMODE_TEXT_40x25 and **P_VMODE_TEXT_40x25_CB** are equivalent.

P_VMODE_TEXT_80x25, **P_VMODE_TEXT_80x25_CB**, and **P_VMODE_TEXT_80x25_MONO** are equivalent.

Parameters

None.

Returns

P_VMODE_TEXT_40X25	40x25 text mode without color burst.
P_VMODE_TEXT_40X25_CB	40x25 text mode.
P_VMODE_TEXT_80X25	80x25 text mode without color burst.
P_VMODE_TEXT_80X25_CB	80x25 text mode.
P_VMODE_GRAPHIC_320X200	320x200 resolution graphics mode without color burst.
P_VMODE_GRAPHIC_320X200_CB	320x200 resolution graphics mode.
P_VMODE_GRAPHIC_640X200	640x200 resolution graphics mode.
P_VMODE_TEXT_80X25_MONO	80x25 text mode, 2 color.

Sample

None.

BIOS_GetViewportMode

p_bios.h

Syntax

```
UCHAR BIOS_GetViewportMode(void);
```

Description

Returns the viewport mode settings which affect text wrapping and panning of the viewport.

Parameters

None.

Returns

Bit flags defined by the following constants. These flags may be combined in any logical combination.

P_VIEWPT_HPAN	Enables horizontal panning.
P_VIEWPT_VPAN	Enables vertical panning.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void ReadViewportPosMode(void)
{
    UINT mode, x, y;
    UINT width, height;
    UCHAR scanCode, ch;

    mode = BIOS_GetViewportMode();
    BIOS_GetViewportPos(&x, &y);
    BIOS_GetViewportSize(&width, &height);
    printf ("Pos : (%u, %u)\nMode: 0x%02x\n", x, y, mode);
    printf ("Horiz. panning is %s, \n", mode&VIEWPT_MODE_HMOVE ?
        "ON" : "OFF");
    printf ("Vert. panning is %s \n", mode&VIEWPT_MODE_VMOVE ?
        "ON" : "OFF");
    printf ("Size : (%u, %u)\n", width, height);
    printf ("\n  < Hit any key >  ");
    while (!BIOS_GetKeyStatus(&scanCode, &ch));
```

```
    BIOS_KeyboardReadChar(&scanCode);
}
```

BIOS_GetViewportPos

p_bios.h

Syntax

```
void BIOS_GetViewportPos(UINT *Row, UINT *Column);
```

Description

Returns the current position of the viewport screen relative to the logical display screen.

If panning is not enabled, the viewport and the physical screen are the same; **Row** and **Column** are both zero.

Parameters

None.

Returns

Row	Top row of logical screen visible on the physical screen.
Column	Leftmost column of logical screen visible on the physical screen.

Sample

See **BIOS_GetViewportMode()**.

BIOS_GetViewportSize

p_bios.h

Syntax

```
void BIOS_GetViewportSize(UINT *Rows, UINT *Columns);
```

Description

Returns the number of rows and columns of the viewport.

Parameters

None.

Returns

Rows	Number of rows that can be seen in the display window. The default (6x8 font) is 8.
Columns	Number of columns that can be seen in the display window. The default (6x8 font) is 21.

Sample

See **BIOS_GetViewportMode()**.

BIOS_IsCursorOn

p_bios.h

Syntax

```
UINT BIOS_IsCursorOn(void);
```

Description

Checks to see whether cursor is visible.

Parameters

None.

Returns

FALSE	Cursor is not visible.
TRUE	Cursor is visible.

Sample

See **BIOS_CursorOff()**.

BIOS_KeyboardReadChar

p_bios.h

Syntax

```
UCHAR BIOS_KeyboardReadChar(UCHAR *ScanCode);
```

Description

Returns keyboard scan code and ASCII character from the keyboard buffer. **`BIOS_KeyboardReadChar()`** removes the keystroke from the keyboard buffer. To retrieve this information without removing the keystroke, use **`BIOS_GetKeyStatus()`**.

Parameters

None.

Returns

`ScanCode`	Scan code of the key pressed.
`Character`	ASCII value of key pressed.

Sample

See **`BIOS_APMGetStatus`**.

BIOS_KeyboardWriteChar

p_bios.h

Syntax

```
int BIOS_KeyboardWriteChar(UCHAR ScanCode, UCHAR Ch);
```

Description

Inserts a keyboard scan code and ASCII character into the keyboard buffer. The new character appears after any other keystrokes which may already be in the keyboard buffer.

Parameters

ScanCode	Scan code of the key pressed.
Ch	ASCII value of key pressed.

Returns

FALSE	Keystroke successfully placed in keyboard buffer.
TRUE	Keyboard buffer is full.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void IntoKeyboardBuffer(void)
{
    UCHAR scanCode, ch;

    // Put a character into the keyboard buffer
    BIOS_KeyboardWriteChar (30, 'a');

    // Read it back out and display
    ch = BIOS_KeyboardReadChar (&scanCode);
    printf ("Char %c Scan %u\n", ch, scanCode);
    return;
}
```

BIOS_Load6x8Font

p_bios.h

Syntax

```
void BIOS_Load6x8Font(void);
```

Description

Loads the internal 6x8 font as the new active font.

Parameters

None.

Returns

None.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void PickAFont(void)
{
    UCHAR scanCode, ch;
    int   done = 0;

    BIOS_ClearScreen(P_ATT_NORM);
    printf ("1. Load 6x8\n");
    printf ("2. Load 8x8\n");
    printf ("3. Load 8x16\n");
    printf ("ESC: Exit");

    // Put a character into the keyboard buffer
    while (!done)
    {
        while (!BIOS_GetKeyStatus (&scanCode, &ch));
        ch = BIOS_KeyboardReadChar (&scanCode);
        switch (ch)
        {
            case '1': BIOS_Load6x8Font(); break;
            case '2': BIOS_Load8x8Font(); break;
            case '3': BIOS_Load8x16Font(); break;
            case 27: done = 1; break;
            default: break;
```

```
        }
    }
    return;
}
```

BIOS_Load8x16Font

p_bios.h

Syntax

```
void BIOS_Load8x16Font(void);
```

Description

Loads the internal 8x16 font as the new active font.

Parameters

None.

Returns

None.

Sample

See **BIOS_Load6x8Font()**.

BIOS_Load8x8Font

p_bios.h

Syntax

```
void BIOS_Load8x8Font(void);
```

Description

Loads the internal 8x8 font as the new active font.

Parameters

None.

Returns

None.

Sample

See **`BIOS_Load6x8Font()`**.

BIOS_LoadUserFont

p_bios.h

Syntax

```
void BIOS_LoadUserFont(UINT CharWidth, UINT CharHeight, UCHAR
far *FontTable);
```

Description

Loads a user-defined font table as the new active font. The font table should be organized into 256 consecutive entries, with each entry consisting of the bytes that make up the bitmap for a character. Each byte comprises a single scan line in the bitmap. Only a pointer to the font table is retained by the BIOS, and therefore no steps are taken to ensure that the font table is intact when it is used. The active font may be changed only while in a text mode **(P_VMODE_TEXT_...)**.

Parameters

CharWidth Pixel width for each character. This is limited to 6 or 8.
CharHeight Pixel height for each character. This is limited to 8 or 16.
FontTable Address of font table.

Returns

None.

Sample

None.

BIOS_MoveViewportDown

p_bios.h

Syntax

```
void BIOS_MoveViewportDown(UCHAR Rows);
```

Description

Moves the viewport down by the specified number of rows. The top line(s) of the display scrolls off the screen; more line(s) become visible at the bottom. Vertical panning must be enabled using **`BIOS_SetViewportMode()`** for this function to have any effect. A call to **`BIOS_MoveViewportDown()`** when the viewport is at the bottom of the logical screen has no effect.

Parameters

`Rows` Number of rows to move viewport.

Returns

None.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void MoveViewport()
{
    char ch;
    puts("Move the viewport \nwith IJKL keys. \n"
        Press ESC to stop.");
    while (1)
    {
        ch = getch();
        switch (toupper(ch))
        {
            case 27:    return;
            case 'I':   BIOS_MoveViewportUp(1); break;
            case 'K':   BIOS_MoveViewportDown(1); break;
            case 'J':   BIOS_MoveViewportLeft(1); break;
            case 'L':   BIOS_MoveViewportRight(1); break;
        }
    }
}
```

BIOS_MoveViewportLeft

p_bios.h

Syntax

```
void BIOS_MoveViewportLeft(UCHAR Columns);
```

Description

Moves the viewport left by the specified number of columns. The rightmost column(s) of the display scrolls off the screen; more column(s) become visible at the left. Horizontal panning must be enabled using **BIOS_SetViewportMode()** for this function to have any effect. A call to **BIOS_MoveViewportLeft()** when the viewport is at the left of the logical screen has no effect.

Parameters

Columns Number of columns to move viewport.

Returns

None.

Sample

See **BIOS_MoveViewportDown()**.

BIOS_MoveViewportRight

p_bios.h

Syntax

```
void BIOS_MoveViewportRight(UCHAR Columns);
```

Description

Moves the viewport right by the specified number of columns. The leftmost column(s) of the display scrolls off the screen; more column(s) become visible at the right. Horizontal panning must be enabled using **`BIOS_SetViewportMode()`** for this function to have any effect. A call to **`BIOS_MoveViewportRight()`** when the viewport is at the right of the logical screen has no effect.

Parameters

`Columns` Number of columns to move viewport.

Returns

None.

Sample

See **`BIOS_MoveViewportDown()`**.

BIOS_MoveViewportUp

p_bios.h

Syntax

```
void BIOS_MoveViewportUp(UCHAR Rows);
```

Description

Moves the viewport up by the specified number of rows. The bottom line(s) of the display scrolls off the screen; more line(s) become visible at the top. Vertical panning must be enabled using **BIOS_SetViewportMode()** for this function to have any effect. A call to **BIOS_MoveViewportUp()** when the viewport is at the top of the logical screen has no effect.

Parameters

Rows Number of rows to move viewport.

Returns

None.

Sample

See **BIOS_MoveViewportDown()**.

BIOS_ReadPixel

p_bios.h

Syntax

```
UCHAR BIOS_ReadPixel(UINT DisplayPage, UINT Row, UINT Column);
```

Description

Returns the pixel color at the specified coordinates on the specified display page. Pixels may be read only in one of the graphics modes **(P_VMODE_GRAPHICS_...)**.

Parameters

DisplayPage Video display page of cursor to check.
P_ACTIVE_PAGE indicates the current active display page.

Row Graphics *x* coordinate for pixel to read.

Column Graphics *y* coordinate for pixel to read.

Returns

Pixel Color The unit has no color display capability. However, if a color is used to write a pixel, that color code is preserved and returned by this function.

Sample

None.

BIOS_RefreshDisplay

p_bios.h

Syntax

```
void BIOS_RefreshDisplay(void);
```

Description

Forces the display to be redrawn based on the contents of video RAM. The unit's video memory is nonstandard in two ways: it is not located at the expected memory range beginning at 0B0000h and it is not used by hardware to automatically refresh the screen. The memory range on the unit is from 0A0000h to 0A4000h. This function provides a manual refresh to replace the typical hardware refresh available on PCs.

Parameters

None.

Returns

None.

Sample

None.

BIOS_ScrollPageDown

p_bios.h

Syntax

```
void BIOS_ScrollPageDown(UINT Lines, UINT Attrib, UINT TRow,
UINT LCol, UINT BRow, UINT RCol);
```

Description

Scrolls the specified window on the active video page down. The area within the specified window is scrolled down by the requested number of lines. Text that is scrolled beyond the bottom of the window is lost. The new lines that appear at the top of the window are filled with spaces carrying the **Attrib** specified.

Parameters

Lines	Number of lines to scroll window.
Attrib	Screen attribute that is to be written to the screen as the screen is being scrolled. **P_ATT_NORM** specifies a normal white-on-black display attribute. **P_ATT_INVERSE** specifies a reverse-video (black-on-white) display attribute.
TRow	Top line of window to scroll.
LCol	Left column of window to scroll.
BRow	Bottom line of window to scroll.
RCol	Right column of window to scroll.

Returns

None.

Sample

```
void AlternateClearScreen(void)
{
    UCHAR scanCode, ch;

    // You can clear the screen with the scrolling
    // functions.
    BIOS_ScrollPageDown (25, P_ATT_NORM, 0, 0, 24, 79);
    while (!BIOS_GetKeyStatus(&scanCode, &ch));
    BIOS_KeyboardReadChar(&scanCode);
}
```

BIOS_ScrollPageUp

p_bios.h

Syntax

```
void BIOS_ScrollPageUp(UINT Lines, UINT Attrib, UINT TRow, UINT
LCol, UINT BRow, UINT RCol);
```

Description

Scrolls the specified window on the active video page up. The area within the specified window is scrolled up by the requested number of lines. Text that is scrolled beyond the top of the window is lost. The new lines that appear at the bottom of the window are filled with spaces carrying the **Attrib** specified.

Parameters

Lines	Number of lines to scroll window.
Attrib	Screen attribute that is to be written to the screen as the screen is being scrolled. **P_ATT_NORM** specifies a normal white-on-black display attribute. **P_ATT_INVERSE** specifies a reverse-video (black-on-white) display attribute.
TRow	Top line of window to scroll.
LCol	Left column of window to scroll.
BRow	Bottom line of window to scroll.
RCol	Right column of window to scroll.

Returns

None.

Sample

See **BIOS_ScrollPageDown()**.

BIOS_ScrollViewportDown

p_bios.h

Syntax

```
void BIOS_ScrollViewportDown(UINT Lines,UINT Attrib);
```

Description

Scrolls the window defined by the viewport down. The area within the physical display is scrolled down by the requested number of lines. Text that is scrolled beyond the bottom of the display is lost. The new lines that appear at the top of the display are filled with spaces carrying the **Attrib** specified.

Parameters

Lines Number of lines to scroll window.

Attrib Screen attribute that is to be written to the screen as the screen is being scrolled. **P_ATT_NORM** specifies a normal white-on-black display attribute. **P_ATT_INVERSE** specifies a reverse-video (black-on-white) display attribute.

Returns

None.

Sample

```
void AlternateClearViewport(void)
{
    UCHAR scanCode, ch;

    // You can clear the screen with the scrolling
    // functions.
    BIOS_ScrollViewportDown (25, P_ATT_NORM, 0, 0, 7, 21);
    while (!BIOS_GetKeyStatus(&scanCode, &ch));
    BIOS_KeyboardReadChar(&scanCode);
}
```

BIOS_ScrollViewportUp

p_bios.h

Syntax

```
void BIOS_ScrollViewportUp(UINT Lines, UINT Attrib);
```

Description

Scrolls the window defined by the viewport up. The area within the physical display is scrolled up by the requested number of lines. Text that is scrolled beyond the top of the display is lost. The new lines that appear at the bottom of the display are filled with spaces carrying the **Attrib** specified.

Parameters

Lines	Number of lines to scroll window.
Attrib	Screen attribute that is to be written to the screen as the screen is being scrolled. **P_ATT_NORM** specifies a normal white-on-black display attribute. **P_ATT_INVERSE** specifies a reverse-video (black-on-white) display attribute.

Returns

None.

Sample

See **BIOS_ScrollViewportDown()**.

BIOS_SetCursorPos

p_bios.h

Syntax

```
void BIOS_SetCursorPos(UINT DisplayPage, UINT Row, UINT Column);
```

Description

Sets the position of the cursor on a specified display page.

Parameters

DisplayPage	Video display page of cursor to set. **P_ACTIVE_PAGE** indicates the current active display page.
Row	Row position of cursor.
Column	Column position of cursor.

Returns

None.

Sample

See **BIOS_VideoReadCharAttr()**.

BIOS_SetCursorStyle

p_bios.h

Syntax

```
void BIOS_SetCursorStyle(UINT StyleCode);
```

Description

Sets the cursor style to one of five available styles. When using 8x16 fonts, cursor emulation must be enabled using **BIOS_EnableCursorEmulation()** for the styles to be effective.

Parameters

StyleCode Indicates the cursor style.

P_CUR_NORM results in underscore. Equivalent to **BIOS_SetCursorType(6,7)**.

P_CUR_FULL results in full cursor. Equivalent to **BIOS_SetCursorType(0,7)**.

P_CUR_HTOP results in half cursor on top. Equivalent to **BIOS_SetCursorType(0,3)**.

P_CUR_HALF results in half cursor on bottom. Equivalent to **BIOS_SetCursorType(3,7)**.

P_CUR_OFF results in no cursor. Equivalent to **BIOS_SetCursorType(1,0)**.

Returns

None.

Sample

See **BIOS_CursorOff()**.

BIOS_SetCursorType

p_bios.h

Syntax

```
void BIOS_SetCursorType(UINT StartLine, UINT EndLine);
```

Description

Sets the cursor type using start and end lines to determine cursor size and positioning.

For text modes with 6x8 or 8x8 fonts, cursor emulation converts the cursor to one of the defined cursor sizes **(P_CURR_NORM, P_CURR_HALF, P_CURR_HTOP, P_CURR_FULL)**.

Use **BIOS_DisableCursorEmulation()** to indicate that **StartLine** and **Endline** should be used without conversion.

Parameters

StartLine	Indicates the top line of the cursor.
EndLine	Indicates the bottom line of the cursor.

Returns

None.

Sample

See **BIOS_CursorOff(), BIOS_DisableCursorEmulation()**.

BIOS_SetCurXY

p_bios.h

Syntax

```
void BIOS_SetCurXY(UINT Row, UINT Column);
```

Description

Sets the position of the cursor on the active display page. This is a macro equivalent to **`BIOS_SetCursorPos(P_ACTIVE_PAGE, Row, Column)`**.

Parameters

`Row`	Row position of cursor.
`Column`	Column position of cursor.

Returns

None.

Sample

See **`BIOS_ClearScreen()`**.

BIOS_SetDate

p_bios.h

Syntax

```
void BIOS_SetDate(UINT Century, UINT Year, UINT Month, UINT Day);
```

Description

Sets the current system date.

Parameters

`Century`	Century (19 or 20).
`Year`	Year (0–99).
`Month`	Month (1–12).
`Day`	Day (1–31).

Returns

None.

Sample

See **`BIOS_GetDate()`**.

BIOS_SetDisplayPage

p_bios.h

Syntax

```
void BIOS_SetDisplayPage(UINT DisplayPage);
```

Description

Sets the currently active display page.

Parameters

DisplayPage Active display page. The range of values is dependent upon video mode.

Returns

None.

Sample

None.

BIOS_SetKbdCtrl

p_bios.h

Syntax

```
void BIOS_SetKbdCtrl(UCHAR Features);
```

Description

Sets the status of the keyboard control flags. These flags enable or disable special key sequences.

Parameters

`Features` Bit flags defined by the following constants. These flags may be combined in any logical combination.

`P_KBDCTRL_PRGM` enables trigger programming.

`P_KBDCTRL_REBOOT` enables **CTRL-ALT-DEL** key sequence.

Returns

None.

Sample

See **`BIOS_GetKbdCtrl()`**.

BIOS_SetKeyClick

p_bios.h

Syntax

```
void BIOS_SetKeyClick(UCHAR KeyClick);
```

Description

Sets the keyboard click setting.

Parameters

KeyClick Keyboard sound setting.
P_KBDSOUND_OFF generates no sound.
P_KBDSOUND_CLICK generates mechanical click.
P_KBDSOUND_BEEP generates beep.

Returns

None.

Sample

See **BIOS_GetKbdCtrl()**.

BIOS_SetKeyInputState

p_bios.h

Syntax

```
void BIOS_SetKeyInputState(UCHAR State);
```

Description

Sets the status of the **ALPHA**, **CAPS LOCK**, **ALT**, **CTRL**, **FN 1**, **FN 2**, **INTL**, and **PRGM** keys.

Parameters

`State` Bit flags defined by the following constants. These flags may be combined in any logical combination.

`P_KBDSTATE_INTL` puts unit in international character entry mode.

`P_KBDSTATE_PRGM` puts unit in programming mode.

`P_KBDSTATE_ALPHA` puts unit in alpha mode.

`P_KBDSTATE_CAPS` makes **CAPS LOCK** active.

`P_KBDSTATE_ALT` makes **ALT** active.

`P_KBDSTATE_CTRL` makes **CTRL** active.

`P_KBDSTATE_FUNC2` makes **FN 2** active.

`P_KBDSTATE_FUNC1` makes **FN 1** active.

Returns

None.

Sample

See **`BIOS_GetKeyInputState()`**.

BIOS_SetSpeakerAlternate

p_bios.h

Syntax

```
void BIOS_SetSpeakerAlternate(UCHAR Index);
```

Description

Sets the alternate speaker tone index.

Parameters

Index	Index into alternate speaker tone table. Valid settings are 0–7, with 0 being the lowest frequency and 7 being the highest.

Returns

None.

Sample

See **BIOS_GetKbdCtrl()**.

BIOS_SetSpeakerVolume

p_bios.h

Syntax

```
void BIOS_SetSpeakerVolume(UCHAR Volume);
```

Description

Sets the speaker volume.

Parameters

`Volume` Volume level.
`P_SNDVOL_OFF` turns speaker off.
`P_SNDVOL_UP` increases volume level.
`P_SNDVOL_DOWN` decreases volume level.
1–7, with 1 being the softest and 7 being the loudest.

Returns

None.

Sample

See **`BIOS_GetKbdCtrl()`**.

BIOS_SetTickCount

p_bios.h

Syntax

```
void BIOS_SetTickCount(ULONG Count);
```

Description

Sets the value in the clock-tick counter. The value is retrieved using **`BIOS_GetTickCount()`**.

Parameters

`Count` Number of clock ticks.

Returns

None.

Sample

None.

BIOS_SetTime

p_bios.h

Syntax

```
void BIOS_SetTime(UINT Hours, UINT Minutes, UINT Seconds, UINT
DST);
```

Description

Sets the system time.

Parameters

Hours	Hours (0–23).
Minutes	Minutes (0–59).
Seconds	Seconds (0–59).
DST	Daylight savings time flag. FALSE (0) if not daylight savings time; TRUE (1) if it is daylight savings time.

Returns

None.

Sample

See **BIOS_GetDate()**.

BIOS_SetVideoMode

p_bios.h

Syntax

```
void BIOS_SetVideoMode(UINT VideoMode);
```

Description

Sets the current video mode. The default video mode is **P_VMODE_TEXT_80x25_CB**.

Some video modes are provided for compatibility only.

P_VMODE_TEXT_40x25 and **P_VMODE_TEXT_40x25_CB** are equivalent.

P_VMODE_TEXT_80x25, **P_VMODE_TEXT_80x25_CB**, and **P_VMODE_TEXT_80x25_MONO** are equivalent.

Parameters

VideoMode Video mode indicated by one of the following constants:

P_VMODE_TEXT_40X25 is 40x25 text mode without color burst.

P_VMODE_TEXT_40X25_CB is 40x25 text mode.

P_VMODE_TEXT_80X25 is 80x25 text mode without color burst.

P_VMODE_TEXT_80X25_CB is 80x25 text mode.

P_VMODE_GRAPHIC_320X200 is 320x200 resolution graphics mode without color burst.

P_VMODE_GRAPHIC_320X200_CB is 320x200 resolution graphics mode.

P_VMODE_GRAPHIC_640X200 is 640x200 resolution graphics mode.

P_VMODE_TEXT_80X25_MONO is 80x25 text mode, two color.

Returns

None.

Sample

None.

BIOS_SetViewportMode

p_bios.h

Syntax

```
void BIOS_SetViewportMode(UINT Mode);
```

Description

Sets the viewport mode for horizontal and vertical panning.

Parameters

Mode Bit flag defined by the following constants:
P_VIEWPT_HPAN enables horizontal panning.
P_VIEWPT_VPAN enables vertical panning.

Returns

None.

Sample

See **BIOS_ClearScreen()**.

BIOS_SetViewportPos

p_bios.h

Syntax

```
void BIOS_SetViewportPos(UINT Row, UINT Column);
```

Description

Moves the viewport to the specified cursor position on the logical screen.

Parameters

Row	Top row of logical screen visible on the physical screen.
Column	Leftmost column of logical screen visible on the physical screen.

Returns

None.

Sample

None.

BIOS_SpeakerAlternate

p_bios.h

Syntax

```
void BIOS_SpeakerAlternate(UINT Duration);
```

Description

Sounds the alternate speaker tone for a specified duration. The tone used is set using the **`BIOS_SetSpeakerAlternate()`**.

Parameters

`Duration` Length of time to sound speaker. This value is designated in milliseconds.

Returns

None.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void SoundSpeaker(void)
{
    UCHAR ScanCode;

    BIOS_ClearScreen(P_ATT_NORM);
    BIOS_SetKeyClick(P_KBDSOUND_OFF);

    printf ("Press any key to\n");
    printf ("sound alternate\n");
    printf ("tone for 500ms.\n");
    BIOS_KeyboardReadChar(&ScanCode);

    BIOS_SpeakerAlternate(500);

    printf ("Press any key to\n");
    printf ("sound speaker\n");
    BIOS_KeyboardReadChar(&ScanCode);

    BIOS_SpeakerTone (2000, 100);
    BIOS_SpeakerTone (2200, 100);
    BIOS_SpeakerTone (2400, 100);
```

```
    BIOS_SpeakerTone (2600, 100);
    return;
}
```

BIOS_SpeakerTone

p_bios.h

Syntax

```
void BIOS_SpeakerTone(ULONG Frequency, UINT Duration);
```

Description

Sounds the speaker at a specified frequency for a specified duration.

Parameters

`Frequency` Frequency in Hertz. Valid values are 1 to 1193180 (1.19318 MHz.). Using a frequency of 0 (zero) has no effect.

`Duration` Length of time to sound speaker. This value is designated in milliseconds.

Returns

None.

Sample

See **`BIOS_SpeakerAlternate()`**.

BIOS_VideoReadCharAtt

p_bios.h

Syntax

```
void BIOS_VideoReadCharAtt(UINT DisplayPage, UCHAR *Attrib,
UCHAR *Ch);
```

Description

Returns the character and its attribute at the current cursor position.

Parameters

DisplayPage Video display page of character to read.
P_ACTIVE_PAGE indicates the current active display page.

Returns

Attrib	Screen attribute of character read. **P_ATT_NORM** is normal white-on-black display attribute. **P_ATT_INVERSE** is reverse-video (black-on-white) display attribute.
Ch	Character read from current position.

Sample

```
#include <stdio.h>
#include "p_bios.h"

void StringAttr()
{
    char  Str[13] = "Hello World!";
    char  AttribStr[26];

    int   index;
    UINT oldx, oldy;
    UCHAR scanCode, ch;
    UCHAR attrib = P_ATT_NORM;

    BIOS_ClearScreen (P_ATT_NORM);

    for (index=0; index<sizeof(Str); index++)
    {
        // Display attribute string. Attribute toggles between
```

```
        // normal and reverse video for each character.
        BIOS_VideoWriteCharAtt (P_ACTIVE_PAGE, attrib, Str[index], 1);
        BIOS_GetCursorPos (P_ACTIVE_PAGE, &oldx, &oldy);

        // Read character from screen and toggle attribute.
        BIOS_VideoReadCharAtt (P_ACTIVE_PAGE, &attrib, &ch);
        BIOS_SetCursorPos (P_ACTIVE_PAGE, oldx, oldy+1);

        AttribStr[index*2] = ch;
        if (attrib == P_ATT_NORM)
            attrib = P_ATT_INVERSE;
        else
            attrib = P_ATT_NORM;
        AttribStr[index*2+1] = attrib;
    }

    // Write new string to 2nd and 3rd line of display
    // Final cursor should be at the end of the second line, since
    // the BIOS_WriteAttribStr does not move the cursor.
    BIOS_WriteAttribStrWithCur (P_ACTIVE_PAGE, sizeof(Str), 1, 0,
        AttribStr);
    BIOS_WriteAttribStr (P_ACTIVE_PAGE, sizeof(Str), 2, 0, AttribStr);
    while (!BIOS_GetKeyStatus(&scanCode, &ch));
    BIOS_KeyboardReadChar(&scanCode);

    return;
}
```

BIOS_VideoWriteCharAtt

p_bios.h

Syntax

```
void BIOS_VideoWriteCharAtt(UINT DisplayPage, UCHAR Attrib,
UCHAR Ch, UINT Count);
```

Description

Writes character(s) to the current cursor position. The cursor is not moved by this function.

Parameters

DisplayPage	Video display page of character to read. **P_ACTIVE_PAGE** indicates the current active display page.
Attrib	Screen attribute to use when writing the character. **P_ATT_NORM** can be used to specify a normal white-on-black display attribute. **P_ATT_INVERSE** can be used to specify a reverse-video (black-on-white) display attribute.
Ch	Character to write.
Count	Number of times to repeat character write.

Returns

None.

Sample

See **BIOS_VideoReadCharAtt()**.

BIOS_VideoWriteCharOnly

p_bios.h

Syntax

```
void BIOS_VideoWriteCharOnly(UINT DisplayPage, UCHAR Ch, UINT
Count);
```

Description

Writes character(s) to the current cursor position. The cursor is not moved by this function.

Parameters

DisplayPage	Video display page of character to read. **P_ACTIVE_PAGE** indicates the current active display page.
Ch	Character to write.
Count	Number of times to repeat character write.

Returns

None.

Sample

None.

BIOS_WriteAttribStr

p_bios.h

Syntax

```
void BIOS_WriteAttribStr(UINT DisplayPage, UINT StrLen, UINT
Row, UINT Column, char far *String);
```

Description

Writes a string to the display. The string contains alternating characters and attribute bytes. The cursor position is not updated after the write.

Parameters

DisplayPage	Video display page where string will be written. **P_ACTIVE_PAGE** indicates the current active display page.
StrLen	Length of string to display.
Row	Row to write string.
Column	Column to write string.
String	Address of string.

Returns

None.

Sample

See **BIOS_VideoReadCharAtt()**.

BIOS_WriteAttribStrWithCur

p_bios.h

Syntax

```
void BIOS_WriteAttribStrWithCur(UINT DisplayPage, UINT StrLen,
UINT Row, UINT Column, char far *String);
```

Description

Writes a string to the display. The string contains alternating characters and attribute bytes. The cursor position is updated after the write.

Parameters

DisplayPage	Video display page where string will be written. **P_ACTIVE_PAGE** indicates the current active display page.
StrLen	Length of string to display.
Row	Row to write string.
Column	Column to write string.
String	Address of string.

Returns

None.

Sample

See **BIOS_VideoReadCharAtt()**.

BIOS_WriteCharTeleMode

p_bios.h

Syntax

```
void BIOS_WriteCharTeleMode(UINT DisplayPage, UCHAR Ch, UINT
Color);
```

Description

Writes a character to the current cursor position. The cursor is not moved by this function.

Parameters

DisplayPage	Video display page of character to read. **P_ACTIVE_PAGE** indicates the current active display page.
Ch	Character to write.
Color	Foreground color to use if in graphics mode. While this function accepts any of the standard colors defined for the current video mode, the unit can display only two colors.

Returns

None.

Sample

None.

BIOS_WritePixel

p_bios.h

Syntax

```
void BIOS_WritePixel(UINT DisplayPage, UINT Row, UINT Column,
UINT Color);
```

Description

Sets the pixel color at the specified coordinates on the specified display page. Pixels may be written only in one of the graphics modes **(P_VMODE_GRAPHICS_...)**.

Parameters

DisplayPage	Video display page of pixel to set. **P_ACTIVE_PAGE** indicates the current active display page.
Row	Graphics *x* coordinate for pixel to write.
Column	Graphics *y* coordinate for pixel to write.
Color	Color to use for pixel. While this function accepts any of the standard colors defined for the current video mode, the unit can display only two colors.

Returns

None.

Sample

None.

BIOS_WriteString

p_bios.h

Syntax

```
void BIOS_WriteString(UINT DisplayPage, UINT Attrib, UINT Row,
UINT Column, char far *String);
```

Description

Writes a string to the display page specified. The cursor position is not updated after the write.

Parameters

DisplayPage Video display page of string to write.
P_ACTIVE_PAGE indicates the current active display page.

Attrib Screen attribute to use when writing the string. **P_ATT_NORM** can be used to specify a normal white-on-black display attribute. **P_ATT_INVERSE** can be used to specify a reverse-video (black-on-white) display attribute.

Row Row to write string.

Column Column to write string.

String Address of string.

Returns

None.

Sample

None.

BIOS_WriteStringWithCur

p_bios.h

Syntax

```
void BIOS_WriteStringWithCur(UINT DisplayPage, UINT Attrib, UINT
Row, UINT Column, char far *String);
```

Description

Writes a string to the display page specified. The cursor position is updated after the write.

Parameters

DisplayPage	Video display page of character to read. **P_ACTIVE_PAGE** indicates the current active display page.
Attrib	Screen attribute to use when writing the string. **P_ATT_NORM** can be used to specify a normal white-on-black display attribute. **P_ATT_INVERSE** can be used to specify a reverse-video (black-on-white) display attribute.
Row	Row to write string.
Column	Column to write string.
String	Address of string.

Returns

None.

Sample

None.

BIOS_WriteStrXY

p_bios.h

Syntax

```
void BIOS_WriteStrXY(UINT Row, UINT Column, char far *String);
```

Description

Writes a white-on-black string to the active display page. The cursor position is updated after the write. This is a macro equivalent to **`BIOS_WriteString(P_ACTIVE_PAGE, P_ATT_NORM, Row, Column, String)`**.

Parameters

`Row`	Row to write string.
`Column`	Column to write string.
`String`	Address of string.

Returns

None.

Sample

See **`BIOS_ClearScreen()`**.

CFG_Read

p_cfg.h

Syntax

```
BYTE CFG_Read(WORD prgcode);
```

Description

Reads a configuration setting from the configuration driver (**PARAMS**). The program code is a value from 0 to 255 corresponding to the configuration setting.

Parameters

prgcode Configuration setting to read:

0x00=Code 39 Enable
0x01=Code 39 Minimum length
0x02=Code 39 Maximum length
0x03=Code 39 Enable checksum
0x04=Code 39 Send checksum
0x05=Code 39 Full ASCII mode
0x08=I 2/5 Enable
0x09=I 2/5 Minimum length
0x0A=I 2/5 Maximum length
0x0B=I 2/5 Enable checksum
0x0C=I 2/5 Send checksum
0x0D=I 2/5 Use length 6 and 14 only
0x10=M 2/5 Enable
0x11=M 2/5 Minimum length
0x12=M 2/5 Maximum length
0x13=M 2/5 Enable checksum
0x14=M 2/5 Send checksum
0x15=S 2/5 Enable
0x16=S 2/5 Minimum length
0x17=S 2/5 Maximum length
0x18=S 2/5 Enable checksum

0x19=S 2/5 Send checksum
0x1A=S 2/5 Use 2-bar start/stop
0x1B=Code 11 Enable
0x1C=Code 11 Minimum length
0x1D=Code 11 Maximum length
0x1E=Code 11 Require 2 check digits
0x1F=Code 11 Send check digit(s)
0x20=Codabar/Ames Enable
0x21=Codabar/Ames Minimum length
0x22=Codabar/Ames Maximum length
0x23=Codabar/Ames Send start/stop
0x24=Codabar/Ames Codabar-to-CLSI conversion
0x25=Codabar/Ames Allow wide intercharacter gap
0x26=MSI Enable
0x27=MSI Minimum length
0x28=MSI Maximum length
0x29=MSI Require 2 check digits
0x2A=MSI 2nd check digit MOD 11
0x2B=MSI Send check digit(s)
0x30=UPC-A Enable
0x31=UPC-A Send system digit
0x32=UPC-A Send check digit
0x33=UPC-A Convert UPC-A to EAN-13
0x34=UPC-E Use system digit 0
0x35=UPC-E Use system digit 1
0x36=UPC-E Convert UPC-E to UPC-A
0x37=UPC-E Send system digit
0x38=UPC-E Send check digit
0x39=EAN/JAN Enable EAN-8/JAN-8
0x3A=EAN/JAN EnableEAN-13/JAN-13
0x3B=EAN/JAN Convert EAN-13 to ISBN
0x3C=UPC/EAN/JAN Allow 2-digit extension
0x3D=UPC/EAN/JAN Allow 5-digit extension
0x3E=UPC/EAN/JAN Require extension
0x3F=EAN/JAN Send check digit

0x40=Code 128 Enable
0x41=Code 128 Minimum length
0x42=Code 128 Maximum length
0x43=UCC/EAN 128 Enable
0x52=Labelcode 4/5 Enable
0x53=Labelcode 4/5 Convert
0xB0=Enable/disable laser programming
0xB1=Autoterminator character
0xB2=Auto-off timer
0xB5=Symbology identifiers
0xB8=Good-read tone
0xB9=Number of good-read tones
0xBA=Good read-tone duration
0xBC =Beeper volume
0xBD =Error tone
0xDD =Keyboard click
0xDE =Laser warmup delay
0xE0=Enable Ctrl-Alt-Del
0xE1=Enable trigger programming
0xE2=Backlight auto time-off

Returns

Value of setting.

Sample

See **CFG_Write()**.

CFG_Write

p_cfg.h

Syntax

```
BYTE CFG_Write(char *command);
```

Description

Sends command string to configuration driver (**PARAMS**). The command string must contain the programming start and end sequences (**$+$-** and **EE** respectively). Calling this function is equivalent to scanning programming bar codes.

Parameters

command String to send to **PARAMS** driver.

Returns

Number of bytes sent to driver.

Sample

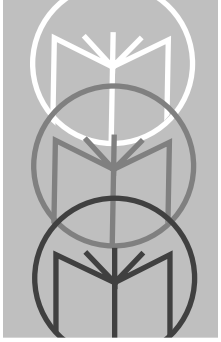
```
#include <stdio.h>
#include "p_cfg.h"

void ScanCode11Label(void)
{
    BYTE Code11Enable;
    char commarg[20];

    // Check to see if Code11 is enabled. Enable it if not.
    Code11Enable = CFG_Read (0x1B);
    if (Code11Enable == FALSE)
        CFG_Write ("$+$-1B1EE");

    //
    // Insert Code11 collection code here.
    //

    // Return to Original Settings
    sprintf (commarg, "$+$-1B%xEE\0", Code11Enable);
    CFG_Write (commarg);
}
```



Chapter 3
Serial Communications

Chapter Contents

This chapter describes the various serial communications function calls.

PORT Data Structure

The first step in accessing a serial port is to open the port by calling the **SIO_Open()** function. **SIO_Open()** returns a pointer to a **PORT** variable loaded with default values. Most of the serial communication functions use the **PORT** data structure described below.

PORT->baseaddr

DOS base address for the communications port in question. For communications port 1 this is usually 3f8h, and for communications port 2 it is 2f8h.

PORT->portid

Communications port number (COM1, COM2, etc.). Currently only communications ports 1 and 2 are supported by SIO. The numbering begins at 0 (i.e., the port ID of COM1 is 0, COM2 is 1, etc.). Use the following constant values when dealing with port numbers:

SIO_COM1	communications port 1.
SIO_COM2	communications port 2.

PORT->databits

The number of data bits the communications port uses to transfer data. Use the following bit constants with **SIO_SetComm()** and **SIO_GetComm()** to set and read this parameter:

SIO_FIVE_DATABITS	Five data bits.
SIO_SIX_DATABITS	Six data bits.
SIO_SEVEN_DATABITS	Seven data bits.
SIO_EIGHT_DATABITS	Eight data bits.

PORT->stopbits

The number of stop bits the communications port uses to transfer data. Use the following constants with **SIO_SetComm()** and **SIO_GetComm()** to set and read this parameter:

SIO_ONE_STOPBIT	One stop bit.
SIO_ONE5_STOPBITS	One and one-half stop bits.

SIO_TWO_STOPBITS Two stop bits.

PORT->parity

The parity the communications port uses to transfer data. The default is no parity. Use the following constants to assign parity values with **SIO_SetComm()**:

SIO_NO_PARITY No parity.
SIO_ODD_PARITY Odd parity.
SIO_EVEN_PARITY Even parity.
SIO_MARK_PARITY Mark parity (parity bit always on).
SIO_SPACE_PARITY Space parity (parity bit always off).

PORT->breakbit

Status of the BREAK signal. This field is TRUE if the communications port in question has asserted the BREAK signal. Otherwise, this bit is FALSE.

PORT->baudrate

The baud rate the communications port uses to transfer data. Use the following constants with **SIO_SetComm()** and **SIO_GetComm()** to set and read this parameter. Note that the baud rate is stored as a **long**.

SIO_BAUD_300 300 baud.
SIO_BAUD_1200 1200 baud.
SIO_BAUD_2400 2400 baud.
SIO_BAUD_4800 4800 baud.
SIO_BAUD_9600 9600 baud.
SIO_BAUD_19K 19200 baud.
SIO_BAUD_38K 38400 baud.
SIO_BAUD_56K 57600 baud.
SIO_BAUD_115K 115200 baud.

PORT->DTR

Status of the Data Terminal Ready (DTR) signal. This field is TRUE if the communications port in question has asserted DTR. Otherwise, this bit is FALSE.

PORT->RTS

Status of the Ready To Send (RTS) signal. This field is TRUE if the communications port in question has asserted the RTS signal. Otherwise, this bit is FALSE.

PORT->GPO1

Status of the General Purpose Output 1 (GPO1) signal. This field is TRUE if the communications port in question has asserted the GPO1 signal. Otherwise, this bit is FALSE.

PORT->GPO2

Status of the General Purpose Output 2 (GPO2) signal. This field is TRUE if the communications port in question has asserted the GPO2 signal. Otherwise, this bit is FALSE. Note that this bit acts as a master interrupt enable signal, and should not be modified if using interrupts.

PORT->CTS

Status of the Clear To Send (CTS) signal. This field is TRUE if the CTS input signal is asserted for the communications port in question. Otherwise, this bit is FALSE.

PORT->DSR

Status of the Data Set Ready (DSR) signal. This field is TRUE if the DSR input signal is asserted for the communications port in question. Otherwise, this bit is FALSE.

PORT->RI

Status of the Ring In (RI) signal. This field is TRUE if the RI input signal is asserted for the communications port in question. Otherwise, this bit is FALSE.

PORT->DCD

Status of the Data Carrier Detect (DCD) signal. This field is TRUE if the DCD input signal is asserted for the communications port in question. Otherwise, this bit is FALSE.

PORT->protocol

The file transfer protocol the communications port uses for **`SIO_FileTransfer()`** API calls. If a protocol routine is called directly, this parameter is ignored. Use the following constants to set and read this parameter:

SIO_PACKNAK	Symbol ACK/NAK protocol.
SIO_XMODEM	Xmodem checksum or CRC protocol with 256 byte blocks.
SIO_ASCIIDUMP	Outputs time-delayed ASCII packets.

PORT->commstatus

The bits in this **WORD** describe the status of the communications port. Never write to this variable, as this makes its contents invalid. The following bit masks can be **AND**ed to **PORT->commstatus** in an **if** statement to ascertain the communications port status. The recommended method is to call **SIO_GetCommError()**, which returns **commstatus** and clears any pending errors.

SIO_ERR_OVERRUN	New characters came into the communications port before existing characters were read. Old characters were discarded. When interrupts are enabled, this means the read buffer is full.
SIO_ERR_PARITY	A parity error occurred on the communications port.
SIO_ERR_FRAMING	A framing error (start/stop bit) occurred.
SIO_ERR_BREAK_DETECT	A break signal was detected on the communications port.
SIO_ERR_WRITEBUFFERFULL	The write buffer is full. With interrupts off, this means the last character sent to the UART hasn't been dispatched. With interrupts on, it indicates the write buffer is full.
SIO_ERR_READBUFFERFULL	The read buffer is full. With interrupts off, this error should never occur. With interrupts on, it indicates the read buffer is full.

PORT->interrupts

The bits in this **BYTE** describe the state of interrupts for the communications port. Writing to this value has no effect on the actual state of the communications port interrupts. If a value is written to this variable, the existing status of the communications port is lost and must be accessed by directly checking the PIC and the UART in question. The following bit masks can be **AND**ed to **PORT->interrupts** in an **if** statement to identify which interrupts are active:

SIO_INTR_NONE	No interrupts are active for this communications port.
SIO_INTR_READ	Read interrupts are active for this communications port.
SIO_INTR_WRITE	Write interrupts are active for this communications port.

PORT->ShowRx()

A pointer to a user-defined function that displays file-receive progress to the user. This function provides a way to embed a progress indicator in the existing file transfer protocols. No method currently exists for communicating information (file size, number of bytes transferred, etc.) between the file-transfer routines and the user function. Generally, this routine is called in the file-transfer routine when one packet is successfully received.

PORT->ShowTx()

A pointer to a user-defined function that displays file-transmit progress to the user. This function provides a way to embed a progress indicator in the existing file-transfer protocols. No method currently exists for communicating information (file size, number of bytes transferred, etc.) between the file-transfer routines and the user function. Generally, this routine is called in the file-transfer routine when one packet is successfully sent and acknowledged.

SIO_AutoClose

sio.h

Syntax

```
void SIO_AutoClose (BOOL bEnable);
```

Description

Enables or disables automatic closing of the communications port upon exit. By default, an open port is closed automatically when a program terminates, either by normal exit or by **CTRL**-**BREAK**. In the default case, **`SIO_Close()`** does not need to be called. If **`AutoClose`** is set to FALSE, **`SIO_Close()`** *must* be called prior to exiting the program. This function must be called prior to calling **`SIO_Open()`**.

Parameters

`bEnable` Indicates whether to perform automatic close. TRUE closes the communications port upon exit. FALSE does not close the communications port on exit.

Returns

None.

Sample

See **`SIO_Open()`**.

SIO_Close

sio.h

Syntax

```
void SIO_Close (PORT *port);
```

Description

Closes the communications port, releases the software "lock" on the port, restores previous port characteristics, resets interrupts if needed, and frees any memory structures that were allocated. This procedure must be called prior to program termination if **`SIO_AutoClose()`** is set to FALSE.

Parameters

`port` Pointer to port structure of port to close.

Returns

None.

Sample

See **`SIO_Open()`**.

SIO_FileTransfer

sio.h

Syntax

```
int SIO_FileTransfer (PORT *port, char szFileName[], FLAGS
flags, PWORD extradata);
```

Description

Provides for automatic uploading and downloading of files. The protocol specified in the port data structure performs the transfer and affects how the parameters are interpreted.

Parameters

`port` Pointer to port structure containing configuration to use for file transfer. See *PORT Data Structure* for the port structure definition.

`szFileName` Path and file name of file to transfer.

`flags` Bit flag defined by the following constants:

`SIO_FLG_TRANSMIT` indicates that **`szFileName`** is transmitted out the communications port.

`SIO_FLG_RECEIVE` indicates that **`szFileName`** is received through the communications port.

`SIO_FLG_APPEND` indicates that data is received and appended to **`szFileName`**. This is available only with the Symbol ACK/NAK protocol.

Note: ASCII dump ignores this parameter and always behaves as if **`SIO_FLG_TRANSMIT`** was set.

extradata A pointer to a value or structure that can be used as a protocol-specific parameter.

Symbol ACK/NAK uses this to designate a transfer timeout value.

ASCII dump uses this to designate a packet delay value.

Xmodem does not use this parameter.

Returns

SIO_ERR_NONE	File transferred successfully.
SIO_ERR_INVALIDFTP	Protocol specified is not valid.
SIO_ERR_TIMEOUT	Transfer timed out.
SIO_ERR_KEYSTROKE	Transfer aborted by user keystroke.
SIO_ERR_FILEOPEN	File could not be opened.
SIO_ERR_FILECREATE	File could not be created.
SIO_ERR_FILEAPPEND	File could not be opened for append.
SIO_ERR_FILEWRITE	An error occurred while trying to write to the file.
SIO_ERR_LINELENGTH	Line in file is too long.
SIO_ERR_MALLOC	Memory could not be allocated for data buffer.
SIO_ERR_PACKETNBR	The packet received was out of order.
SIO_ERR_UNKNOWN	Unknown error.

Sample

See **SIO_Open()**.

SIO_FlushComm

sio.h

Syntax

```
void SIO_FlushComm (PORT *port, FLAG flag);
```

Description

Flushes the serial read and write interrupt buffers. Also clears all incoming serial data. Data waiting to be sent in the UART's transmitter buffer is not cleared.

Parameters

port	Pointer to port structure containing the buffers to clear.
flags	Bit flag defined by the following constants: **SIO_FLUSHREAD** indicates that the read buffer should be cleared. **SIO_FLUSHWRITE** indicates that the write buffer should be cleared. **SIO_FLUSHALL** indicates that both buffers should be cleared.

Returns

None.

Sample

None.

SIO_GetComm

sio.h

Syntax

```
void SIO_GetComm (PORT *port);
```

Description

Retrieve communications port parameters, such as the value of the parity, stop bits, data bits, break bit, RTS, DTR, GPO1, GPO2, CTS, RI, DCD, DSR, and the baud rate. The values are stored in the port data structure. To use this function, call it with the proper port structure as a parameter, and read the desired data from the port structure directly.

Parameters

None.

Returns

port	Pointer to port structure containing configurations returned. See *PORT Data Structure* for the port structure definition.

Sample

None.

SIO_GetCommError

sio.h

Syntax

```
WORD SIO_GetCommError (PORT *port);
```

Description

Returns the contents of **PORT_commstatus** and clears any errors that might have been pending. Although the **PORT_commstatus** variable could be checked manually, the error status wouldn't be cleared, and the action of checking for errors wouldn't be as obvious from a programmer's point of view.

Parameters

port	Pointer to port structure containing **commstatus** to retrieve. See *PORT Data Structure* for the port structure definition.

Returns

SIO_ERR_OVERRUN	New characters came into the communications port before existing characters were read. Old characters were discarded. When interrupts are enabled, this means the read buffer is full.
SIO_ERR_PARITY	A parity error occurred on the communications port.
SIO_ERR_FRAMING	A framing error (start/stop bit) occurred.
SIO_ERR_BREAK_DETECT	A break signal was detected on the communications port.
SIO_ERR_WRITEBUFFERFULL	The write buffer is full. With interrupts off, this means the last character sent to the UART hasn't been dispatched. With interrupts on, it indicates the write buffer is full.
SIO_ERR_READBUFFERFULL	The read buffer is full. With interrupts off, this error should never occur. With interrupts on, it indicates the read buffer is full.

Sample

None.

SIO_Keyflush

sio.h

Syntax

```
void SIO_Keyflush (void);
```

Description

Clears the type-ahead keyboard buffer.

Parameters

None.

Returns

None.

Sample

None.

SIO_Keyhit

sio.h

Syntax

```
BYTE SIO_Keyhit (void);
```

Description

Checks whether a keystroke is available from the keyboard buffer.

Parameters

None.

Returns

FALSE	No character available.
TRUE	At least one character is available.

Sample

None.

SIO_Open

sio.h

Syntax

```
PORT* SIO_Open (BYTE portid, UINT size_rb, UINT size_wb);
```

Description

Initializes the communications port, setting up interrupts (if requested) and setting the port parameters to default values. All necessary memory buffers are allocated here, and various flags are set to indicate that the port is being used.

Parameters

portid	Communications port to open. **SIO_COM1** indicates COM1. **SIO_COM2** indicates COM2.
size_rb	Size of the read buffer. If zero is specified, read interrupts are not enabled for this port.
size_wb	Size of the write buffer. If zero is specified, write interrupts are not enabled for this port.

Returns

Pointer to a port structure that contains the characteristics of the opened port.

Sample

```
#include <stdlib.h>
#include “sio.h”
#include “filexfer.h”

void main (void)
{
    BYTE    FileName[] = “MYDATA.TXT”;    // Define a string to
                                          // transmit.
    PORT    *port = NULL;                 // Define the comm port.

    // Disable automatic port close. The default is TRUE.
    SIO_AutoClose (FALSE);

    // Open Com2 with a read interrupt buffer.
    port = SIO_Open(SIO_COM2, 300, 0);
```

```
    // Setup the comm port for 19,200 baud, XMODEM protocol
    port->baudrate = SIO_BAUD_19K;
    port->protocol = SIO_XMODEM;
    port->RTS = TRUE;
    SIO_SetComm (port);

    // Transfer the file
    SIO_FileTransfer (port, FileName, SIO_FLG_TRANSMIT, 0);

    // Close COM2
    SIO_Close (port);  // Not needed if SIO_AutoClose() hadn't
                       // been called above.
}
```

SIO_Read

sio.h

Syntax

```
int SIO_Read (PORT *port, char szBuffer[], int nlen);
```

Description

Reads the specified number of characters from the specified port.

Parameters

port	Communications port from which to read. See *PORT Data Structure* for the port structure definition.
szBuffer	Character array to store the data read.
nlen	Number of characters to read.

Returns

The number of characters read, or a negative value if an error occurs. If an error occurs, call **SIO_GetCommError()** to determine the nature of the error.

Sample

None.

SIO_SetComm

sio.h

Syntax

```
void SIO_SetComm (PORT *port);
```

Description

Sets the communications port parameters based upon a provided structure.

Parameters

port Communications port structure. See *PORT Data Structure* for the port structure definition.

Returns

None.

Sample

See **SIO_Open()**.

SIO_Ticks

sio.h

Syntax

```
ticks_t SIO_Ticks(void);
```

Description

Gets the number of timer ticks since midnight. There are 18.2 timer ticks every second.

Parameters

None.

Returns

Timer ticks since midnight.

Sample

None.

SIO_Time

sio.h

Syntax

```
DWORD SIO_Time(void);
```

Description

Gets the number of seconds since midnight.

Parameters

None.

Returns

Seconds since midnight.

Sample

None.

SIO_Write

sio.h

Syntax

```
int SIO_Write (PORT *port, char szBuffer[], int nlen);
```

Description

Writes the specified number of characters to the specified port.

Parameters

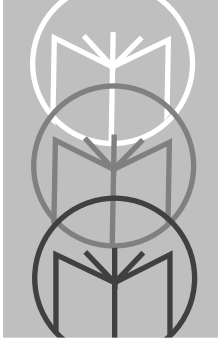
port	Communications port where data is sent.
szBuffer	Character array containing data to be written.
nlen	Number of characters to write.

Returns

The number of characters written, or a negative value if an error occurs. If an error occurs, **SIO_GetCommError()** should be called to determine the nature of the error.

Sample

None.



Chapter 4
A Program Sample

Chapter Contents

This release of the run-time library supports small, medium, and large memory models for both Borland and Microsoft C compilers. The short example that follows uses the sample code from **BIOS_GetKbdCtrl()**. This code is in the file **KEYSAMP.C** and is installed to the **\3200\3200\SAMPLE** directory.

Borland Compilation

The instructions in this section assume the following:

- The Borland C++ compiler is installed to **C:\BC45** with the include files in **C:\BC45\INCLUDE**, the library files in **C:\BC45\LIB**, and **C:\BC45\BIN** in the path.
- The PDT 3200 run-time library is installed to **C:\3200\3200** with the include files in **C:\3200\3200\INCLUDE** and the library files in **C:\3200\3200\LIB**.
- The PDT 3200 code sample is located in **C:\3200\3200\SAMPLE**.

Substitute the appropriate subdirectory paths from your development machine if these assumptions are not correct.

To compile this code sample, complete the following steps:

1. Make the sample directory your current working directory:

   CD\3200\3200\SAMPLE
2. Compile **KEYSAMP.C**:

```
small model:
bcc -c -ms -IC:\BC45\INCLUDE -IC:\3200\3200\INCLUDE
    KEYSAMP.C

medium model:
bcc -c -mm -IC:\BC45\INCLUDE -IC:\3200\3200\INCLUDE
    KEYSAMP.C

large model:
bcc -c -ml -IC:\BC45\INCLUDE -IC:\3200\3200\INCLUDE
    KEYSAMP.C

Compiler Switches:
-c  : Compile only.
```


```
-m# : Specify the memory model.  -ms for small, -mm for
medium,
     -ml for large.
-I  : Include files directory.
```

3. Link **KEYSAMP.OBJ**:

```
small model:
tlink -LC:\BC45\LIB;C:\3200\3200\LIB
     C0S+KEYSAMP,KEYSAMP,,FALCS CS

medium model:
tlink -LC:\BC45\LIB;C:\3200\3200\LIB
     C0M+KEYSAMP,KEYSAMP,,FALCM CM

large model:
tlink -LC:\BC45\LIB;C:\3200\3200\LIB
     C0L+KEYSAMP,KEYSAMP,,FALCL CL

Linker Switches:
-L  : Library search path
```

KEYSAMP.EXE should now exist in the **C:\3200\3200\SAMPLE** subdirectory. You may download it to the PDT 3200 unit to run.

Microsoft Compilation

The instructions in this section assume the following:

- The Microsoft C++ compiler is installed to **C:\MSVC** with the include files in **C:\MSVC\INCLUDE**, the library files in **C:\MSVC\LIB**, and **C:\MSVC\BIN** in the path.
- The PDT 3200 run-time library is installed to **C:\3200\3200** with the include files in **C:\3200\3200\INCLUDE** and the library files in **C:\3200\3200\LIB**.
- The PDT 3200 code sample is located in **C:\3200\3200\SAMPLE**.

Substitute the appropriate subdirectory paths from your development machine if these assumptions are not correct.

To compile this code sample, complete the following steps:

1. Make the sample directory your current working directory:

   ```
   CD\3200\3200\SAMPLE
   ```

2. Compile **KEYSAMP.C**:

   ```
   small model:
   cl /c /AS /IC:\MSVC\INCLUDE /IC:\3200\3200\INCLUDE
       KEYSAMP.C

   medium model:
   cl /c /AM /IC:\MSVC\INCLUDE /IC:\3200\3200\INCLUDE
       KEYSAMP.C

   large model:
   cl /c /AL /IC:\MSVC\INCLUDE /IC:\3200\3200\INCLUDE
       KEYSAMP.C

   Compiler Switches:
   /c  : Compile only.
   /A# : Specify the memory model.  /AS for small, /AM for
          medium, /AL for large.
   /I  : Include files directory.
   ```

3. Link **KEYSAMP.OBJ**:


```
small model:
link KEYSAMP,KEYSAMP,,C:\MSVC\LIB\OLDNAMES.LIB
    C:\MSVC\LIB\SLIBCE.LIB C:\3200\3200\LIB\FALCS.LIB;

medium model:
link KEYSAMP,KEYSAMP,,C:\MSVC\LIB\OLDNAMES.LIB
    C:\MSVC\LIB\MLIBCE.LIB C:\3200\3200\LIB\FALCM.LIB;

large model:
link KEYSAMP,KEYSAMP,,C:\MSVC\LIB\OLDNAMES.LIB
    C:\MSVC\LIB\LLIBCE.LIB C:\3200\3200\LIB\FALCL.LIB;
```

KEYSAMP.EXE should now exist in the **C:\3200\3200\SAMPLE** subdirectory. You may download it to the PDT 3200 unit to run.

Tell Us What You Think...

We'd like to know what you think about this manual. Please take a moment to fill out this questionnaire and fax this form to: (516) 738-3318, or mail to:

Symbol Technologies, Inc.
One Symbol Plaza M/S B-4
Holtsville, NY 11742-1300
Attn: Technical Publications Manager

IMPORTANT: If you need product support, please call the appropriate customer support number provided. Unfortunately, we cannot provide customer support at the fax number above.

User's Manual Title:_____

_____(please include revision level)

How familiar were you with this product before using this manual?

®Very familiar ® Slightly familiar ® Not at all familiar

Did this manual meet your needs? If not, please explain. _____

_____

_____

What topics need to be added to the index, if applicable?_____

_____

What topics do you feel need to be better discussed? Please be specific. _____

_____

_____

What can we do to further improve our manuals? _____

_____

_____

Thank you for your input—We value your comments.