7

console.h

This header file contains procedures and types, which help you port a program that was written for a command-line/console interface to the Macintosh operating system.

The console.h header file consist of various runtime declarations that pertain to the Classic and Carbon Macintosh interfaces.

Overview of console.h

This header file defines the facilities as follows:

- "ccommand" on page 41 helps you port a program that relies on command-line arguments.
- "clrscr" on page 43 clears the SIOUX window and flushes the buffer.
- "getch" on page 43 returns the keyboard character pressed when an ascii key is pressed.
- "InstallConsole" on page 44 installs the Console package.
- "kbhit" on page 44 returns true if any keyboard key is pressed without retrieving the key.
- "ReadCharsFromConsole" on page 45 reads from the Console into a buffer.
- "RemoveConsole" on page 45 removes the console package.
- "__ttyname" on page 46 returns the name of the terminal associated with the file id. The unix.h function ttyname calls this function.
- "WriteCharsToConsole" on page 46 writes a stream of output to the Console window.

ccommand

Lets you enter command-line arguments for a SIOUX program.

```
#include <console.h>
int ccommand(char ***argv);
```

Table 7.1 ccommand

argv	char ***	The address of the second parameter of your command line

Remarks

The function ccommand() must be the first code generated in your program. It must directly follow any variable declarations in the main function.

This function displays a dialog that lets you enter arguments and redirect standard input and output. Please refer to "Overview of SIOUX" on page 255, for information on customizing SIOUX, or setting console options.

Only `stdin, stdout, cin, and cout` are redirected. Standard error reporting methods `stderr, cerr,` and `clog` are not redirected.

The maximum number of arguments that can be entered is determined by the value of MAX_ARGS defined in ccommand.c and is set to 25. Any arguments in excess of this number are ignored.

Enter the command-line arguments in the Argument field. Choose where your program directs standard input and output with the buttons below the field: the buttons on the left are for standard input and the buttons on the right are for standard output. If you choose Console, the program reads from or write to a SIOUX window. If you choose File, `ccommand()` displays a standard file dialog which lets you choose a file to read from or write to. After you choose a file, its name replaces the word *File*.

The function `ccommand()` returns an integer and takes one parameter which is a pointer to an array of strings. It fills the array with the arguments you entered in the dialog and returns the number of arguments you entered. As in UNIX or DOS, the first argument, the argument in element 0, is the name of the program. Listing 7.1 has an example of command line usage.

This function returns the number of arguments you entered.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"Customizing SIOUX" on page 258

Listing 7.1 Example of ccommand() Usage

```
#include <stdio.h>
#include <console.h>
```

```
int main(int argc, char *argv[])
{
  int i;

  argc = ccommand(&argv);

  for (i = 0; i < argc; i++)
    printf("%d. %s\n", i, argv[i]);
  return 0;
}
```

clrscr

Clears the console window and flushes the buffers;

```
#include <console.h>
```

```
void clrscr(void);
```

Remarks

This function is used to select all and clear the screen and buffer by calling SIOUXclrscr from SIOUX.h.

Macintosh only—this function may not be implemented on all Mac OS versions.

getch

Returns the keyboard character pressed when an ascii key is pressed

```
#include <console.h>
```

```
int getch(void);
```

Remarks

This function is used for console style menu selections for immediate actions.

Returns the keyboard character pressed when an ascii key is pressed.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"kbhit" on page 44

InstallConsole

Installs the Console package.

```
#include <console.h>
extern short InstallConsole(short fd);
```

Table 7.2 InstallConsole

fd	short	A file descriptor for standard i/o

Remarks

Installs the Console package, this function will be called right before any read or write to one of the standard streams.

This function returns any error.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"RemoveConsole" on page 45

kbhit

Returns true if any keyboard key is pressed.

```
#include <console.h>
int kbhit(void);
```

Remarks

Returns true if any keyboard key is pressed without retrieving the key used for stopping a loop by pressing any key

This function returns non zero when any keyboard key is pressed.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"getch" on page 43

ReadCharsFromConsole

Reads from the Console into a buffer.

```
#include <console.h>
extern long ReadCharsFromConsole
  (char *buffer, long n);
```

Table 7.3 ReadCharsFromConsole

buffer	char *	A stream buffer
n	long	Number of char to read

Remarks

Reads from the Console into a buffer. This function is called by read.

Any errors encountered are returned.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"WriteCharsToConsole" on page 46

RemoveConsole

Removes the console package.

```
#include <console.h>
extern void RemoveConsole(void);
```

Remarks

Removes the console package. It is called after all other streams are closed and exit functions (installed by either atexit or __atexit) have been called.

Since there is no way to recover from an error, this function doesn't need to return any.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"InstallConsole" on page 44

__ttyname

Returns the name of the terminal associated with the file id.

```
#include <console.h>
extern char *__ttyname(long fildes);
```

Table 7.4 _ttyname

fildes	long	The file descriptor

Remarks

Returns the name of the terminal associated with the file id. The unix.h function ttyname calls this function (we need to map the int to a long for size of int variance).

Returns the name of the terminal associated with the file id.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"ttyname" on page 534

WriteCharsToConsole

Writes a stream of output to the Console window.

```
#include <console.h>
extern long WriteCharsToConsole(char *buffer, long n);
```

Table 7.5 WriteCharsToConsole

buffer	char *	A stream buffer
n	long	Number of char to write

Remarks

Writes a stream of output to the Console window. This function is called by write.

Any errors encountered are returned.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"ReadCharsFromConsole" on page 45

29

SIOUX.h

The SIOUX (Simple Input and Output User eXchange) libraries handle Graphical User Interface issues. Such items as menus, windows, and events are handled so your program doesn't need to for C and C++ programs.

Overview of SIOUX

The following section describes the Macintosh versions of the console emulation interface known as SIOUX. The facilities and structure members for the Standard Input Output User eXchange console interface are "Using SIOUX" and "SIOUX for Macintosh".

- "Using SIOUX" on page 255 is a general description of SIOUX properties.
- "SIOUX for Macintosh" on page 256 explains the (Simple Input and Output User eXchange) library for the Macintosh Operating Systems.

NOTE If you're porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

See Also

"MSL Extras Library Headers" on page 4 for information on `POSIX` naming conventions.

Using SIOUX

Sometimes you need to port a program that was originally written for a command line interface such as DOS or UNIX. Or you need to write a new program quickly and don't have the time to write a complete Graphical User Interface that handles windows, menus, and events.

To help you, CodeWarrior provides you with the SIOUX libraries, which handles all the Graphical User Interface items such as menus, windows, and titles so your program doesn't need to. It creates a window that's much like a dumb terminal or TTY but with scrolling. You can write to it and read from it with the standard C functions and C++ operators, such as printf(), scanf(), getchar(), putchar() and the C++ inserter and extractor operators << and >>. The SIOUX and WinSIOUX

libraries also creates a File menu that lets you save and print the contents of the window. The Macintosh hosted SIOUX includes an Edit menu that lets you cut, copy, and paste the contents in the window. For information on Macintosh redirecting to or from file the stdin, stdout, cout, and cin input output or commandline arguments.

Macintosh only—this function may not be implemented on all Mac OS versions.

See Also

"Overview of console.h" on page 41.

NOTE If you're porting a UNIX or DOS program, you might also need the functions in other UNIX compatibility headers.

SIOUX for Macintosh

SIOUX for Macintosh contains the following segments.

- "Creating a Project with SIOUX" on page 257 shows a running SIOUX program.
- "Customizing SIOUX" on page 258 shows how to customize your SIOUX window.
  - "The SIOUXSettings Structure" on page 259 list structure members that may be set for altering SIOUX's appearance
- "Using SIOUX windows in your own application" on page 264 contains information for using Mac OS facilities with in your SIOUX project.
  - "path2fss" on page 265 a function similar to PBMakeFSSpec.
  - "SIOUXHandleOneEvent" on page 265 allows you to use an even in SIOUX
  - "SIOUXSetTitle" on page 267 allows you to specify a custom title for SIOUX's window

NOTE A **WASTE**© by **Marco Piovanelli** based SIOUX console is available as a pre-release version. This will allow screen output of over 32k characters. All normal SIOUX functions should work but normal pre-release precautions should be taken. Please read all release notes.

The window is a re-sizable, scrolling text window, where your program reads and writes text. It saves up to 32K of your program's text.

With the commands from the Edit menu, you can cut and copy text from the SIOUX window and paste text from other applications into the SIOUX window. With the commands in the File menu, you can print or save the contents of the SIOUX window.

To stop your program at any time, press Command-Period or Control-C. The SIOUX application keeps running so you can edit or save the window's contents. If you want to exit when your program is done or avoid the dialog asking whether to save the window, see "Changing what happens on quit" on page 263

To quit out of the SIOUX application at any time, choose Quit from the File menu. If you haven't saved the contents of the window, the application displays a dialog asking you whether you want to save the contents of the window now. If you want to remove the status line, see "Showing the status line" on page 264.

Creating a Project with SIOUX

To use the SIOUX library, create a project from a project stationery pads that creates an Console style project.

In this chapter, standard input and standard output refer to `stdin`, `stdout`, `cin`, and `cout`. Standard error reporting such as `stderr`, `clog`, and `cerr` is not redirected to a file using `ccommand()`.

If you want only to write to or read from standard input and output, you don't need to call any special functions or include any special header files. When your program refers to standard input or output, the SIOUX library kicks in automatically and creates a SIOUX window for it.

NOTE Remember that functions like `printf()` and `scanf()` use standard input and output even though these symbols do not appear in their parameter lists.

If you want to customize the SIOUX environment, you must #include `SIOUX.h` and modify `SIOUXSettings` before you use standard input or output. As soon as you use one of them, SIOUX creates a window and you cannot modify it. For more information, see "Customizing SIOUX" on page 258.

If you want to use a SIOUX window in a program that has its own event loop, you must modify `SIOUXSettings` and call the function `SIOUXHandleOneEvent()`. For more information, see "Using SIOUX windows in your own application" on page 264.

If you want to add SIOUX to a project you already created, the project must contain certain libraries.

A PPC project must either contain at least these libraries:

- MSL_All_.PPC.Lib
- InterfaceLib
- MathLib

Or at least:

- MSL C.PPC.Lib

- MSL C++.PPC.Lib (for C++)
- MSL_SIOUX_PPC.Lib
- MSL_Runtime_PPC.Lib
- InterfaceLib
- MathLib

A Carbon project must either contain at least these libraries:

- MSL_All_Carbon.Lib
- CarbonLib

Or at least:

- MSL_C_Carbon.Lib
- MSL_C++_Carbon.Lib (for C++)
- MSL_SIOUX_Carbon.Lib
- CarbonLib

A Mach-O project must contain at least these libraries:

- MSL_All_Mach-O.lib
- MSL_SIOUX_Mach-O.lib (to be implemented)

Or at least:

- MSL_C_Mach-O.lib
- MSL_C++_Mach-O.lib (for C++)
- MSL_SIOUX_Mach-O.lib (to be implemented)
- MSL_Runtime_Mach-O.lib

Customizing SIOUX

This following sections describe how you can customize the SIOUX environment by modifying the structure `SIOUXSettings`. SIOUX examines the data fields of `SIOUXSettings` to determine how to create the SIOUX window and environment.

NOTE To customize SIOUX, you must modify `SIOUXSettings` before you call any function that uses standard input or output. If you modify `SIOUXSettings` afterwards, SIOUX does not change its window.

The first three sections, "Changing the font and tabs" on page 261, "Changing the size and location" on page 262, and "Showing the status line" on page 264, describe how to customize the SIOUX window. The next section, "Changing what happens on quit" on page 263, describe how to modify how SIOUX acts when you quit it. The last section,

"Using SIOUX windows in your own application" on page 264, describes how you can use a SIOUX window in your own Macintosh program.

Table 29.1 summarizes what's in the `SIOUXSettings` structure.

Table 29.1 The SIOUXSettings Structure

This field...		Specifies...
char	initializeTB	Whether to initialize the Macintosh toolbox.
char	standalone	Whether to use your own event loop or SIOUX's.
char	setupmenus	Whether to create File and Edit menus for the application.
char	autocloseonquit	Whether to quit the application automatically when your program is done.
char	asktosaveonclose	Query the user whether to save the SIOUX output as a file, when the program is done.
char	showstatusline	Whether to draw the status line in the SIOUX window.
short	tabspaces	If greater than zero, substitute a tab with that number of spaces. If zero, print the tabs.
short	column	The number of characters per line that the SIOUX window will contain.
short	rows	The number of lines of text that the SIOUX window will contain.
short	toppixel	The location of the top of the SIOUX window.
short	leftpixel	The location of the left of the SIOUX window.

Table 29.1 The SIOUXSettings Structure (*continued*)

This field…		Specifies…
short	fontid	The font in the SIOUX window.
short	fontsize	The size of the font in the SIOUX window.
short	stubmode	SIOUX acts like a stubs library
char	usefloatingwindows	(Carbon) use non floating front window
short	fontface	The style of the font in the SIOUX window.
int	sleep	The default value for the `sleep` setting is zero. Zero gets the most speed out of `SIOUX` by telling the system to not give time to other processes during a WaitNextEvent call. A more appropriate setting (that is more friendly to other processes) is to set the `sleep` value to `GetCaretTime()`.

Listing 29.1 contains a small program that customizes a SIOUX window.

Listing 29.1 Example of Customizing a SIOUX Window

```
#include <stdio.h>
#include <sioux.h>
#include <MacTypes.h>
#include <Fonts.h>

int main(void)
{
   short familyID;

   /* Don't exit the program after it runs or ask whether
      to save the window when the program exit */
   SIOUXSettings.autocloseonquit = false;
```

```
    SIOUXSettings.asktosaveonclose = false;

    /* Don't show the status line */
    SIOUXSettings.showstatusline = false;

    /* Make the window large enough to fit 1 line
       of text that contains 12 characters. */
    SIOUXSettings.columns = 12;
    SIOUXSettings.rows = 1;

    /* Place the window's top left corner at (5,40).  */
    SIOUXSettings.toppixel = 40;
    SIOUXSettings.leftpixel = 5;

    /* Set the font to be 48-point, bold, italic Times.   */
    SIOUXSettings.fontsize = 48;
    SIOUXSettings.fontface = bold + italic;
    GetFNum("\ptimes", &familyID);
    SIOUXSettings.fontid = familyID;

    printf("Hello World!");

    return 0;
}
```

Changing the font and tabs

This section describes how to change how SIOUX handles tabs with the field `tabspaces` and how to change the font with the fields `fontid`, `fontsize`, and `fontface`.

NOTE The status line in the SIOUX window writes its messages with the font specified in the fields `fontid`, `fontsize`, and `fontface`. If that font is too large, the status line may be unreadable. You can remove the status line by setting the field `showstatusline` to `false`, as described in "Showing the status line" on page 264.

To change the font in the SIOUX window, set fontid to one of these values defined in the header file Fonts.h:

- `courier where the ID is kFontIDCourier`
- `geneva where the ID is kFontIDGeneva`
- `helvetica where the ID is kFontIDHelvetica`
- `monaco where the ID is kFontIDMonaco`

- `newYork where the ID is kFontIDNewYork`
- `symbol where the ID is kFontIDSymbol`
- `times where the ID is kFontIDTimes`

By default, fontid is monaco.

To change the character style for the font, set fontface to one of these values:

- `normal`
- `bold`
- `italic`
- `underline`
- `outline`
- `shadow`
- `condense`
- `extend`

To combine styles, add them together. For example, to write text that's bold and italic, set `fontface` to `bold + italic`. By default, `fontface` is `normal`.

To change the size of the font, set `fontsize` to the size. By default, `fontsize` is `9`.

The field `tabspaces` controls how SIOUX handles tabs. If `tabspaces` is any number greater than `0`, SIOUX prints that number of spaces required to get to the next tab position instead of a tab. If `tabspaces` is `0`, it prints a tab. In the SIOUX window, a tab looks like a single space, so if you are printing a table, you should set `tabspaces` to an appropriate number, such as `4` or `8`. By default, `tabspaces` is `4`.

The sample below sets the font to 12-point, bold, italic New York and substitutes 4 spaces for every tab:

```
SIOUXSettings.fontsize = 12;
SIOUXSettings.fontface = bold + italic;
SIOUXSettings.fontid = kFontIDNewYork;
SIOUXSettings.tabspaces = 4;
```

Changing the size and location

SIOUX lets you change the size and location of the SIOUX window.

To change the size of the window, set `rows` to the number of lines of text in the window and set `columns` to the number of characters in each line. SIOUX checks the font you specified in `fontid`, `fontsize`, and `fontface` and creates a window that will be large enough to contain the number of lines and characters you specified. If the window is

too large to fit on your monitor, SIOUX creates a window only as large as the monitor can contain.

For example, the code below creates a window that contains 10 lines with 40 characters per line:

SIOUXSettings.rows = 10;

SIOUXSettings.columns = 40;

By default, the SIOUX window contains 24 rows with 80 characters per row.

To change the position of the SIOUX window, set `toppixel` and `leftpixel` to the point where you want the top left corner of the SIOUX window to be. By setting `toppixel` to `38` and `leftpixel` to 0, you can place the window as far left as possible and just under the menu bar. Notice that if `toppixel` is less than `38`, the SIOUX window is under the menu bar. If `toppixel` and `leftpixel` are both 0, SIOUX doesn't place the window at that point but instead centers it on the monitor.

For example, the code below places the window just under the menu bar and near the left edge of the monitor:

```
SIOUXSettings.toppixel = 40;
SIOUXSettings.leftpixel = 5;
```

Changing what happens on quit

The fields `autocloseonquit` and `asktosaveonclose` let you control what SIOUX does when your program is over and SIOUX closes its window.

The field `autocloseonquit` determines what SIOUX does when your program has finished running. If `autocloseonquit` is `true`, SIOUX automatically exits. If `autocloseonquit` is `false`, SIOUX continues to run, and you must choose Quit from the File menu to exit. By default, `autocloseonquit` is `false`.

NOTE You can save the contents of the SIOUX window at any time by choosing Save from the File menu.

The field `asktosaveonclose` determines what SIOUX does when it exits. If `asktosaveonclose` is `true`, SIOUX displays a dialog asking whether you want to save the contents of the SIOUX window. If `asktosaveonclose` is `false`, SIOUX exits without displaying the dialog. By default, `asktosaveonclose` is `true`.

For example, the code below quits the SIOUX application as soon as your program is done and doesn't ask you to save the output:

```
SIOUXSettings.autocloseonquit = true;
SIOUXSettings.asktosaveonclose = false;
```

Showing the status line

The field `showstatusline` lets you control whether the SIOUX window displays a status line, which contains such information as whether the program is running, handling output, or waiting for input. If `showstatusline` is `true`, the status line is displayed. If `showstatusline` is `false`, the status line is not displayed. By default, `showstatusline` is `false`.

Using SIOUX windows in your own application

This section explains how you can limit how much SIOUX controls your program. But first, you need to understand how SIOUX works with your program. You can consider the SIOUX environment to be an application that calls your `main()` function as just another function. Before SIOUX calls `main()`, it performs some initialization to set up the Macintosh Toolbox and its menu. After `main()` completes, SIOUX cleans up what it created. Even while `main()` is running, SIOUX sneaks in whenever it performs input or output, acting on any menu you've chosen or command key you've pressed.

However, SIOUX lets you choose how much work it does for you. You can choose to handle your own events, set up your own menus, and initialize the Macintosh Toolbox yourself.

When you want to write an application that handles its own events and uses SIOUX windows for easy input and output, set the field `standalone` to `false` before you use standard input or output. SIOUX doesn't use its event loop and sets the field `autocloseonquite` to `true` for you, so the application exits as soon as your program is done. In your event loop, you need to call the function `SIOUXHandleOneEvent()`, described on "Using SIOUX windows in your own application" on page 264.

When you don't want to use SIOUX's menus, set the field `setupmenus` to `false`. If `standalone` is also `false`, you won't be able to create menus, and your program will have none. If `standalone` is `true`, you can create and handle your own menus.

When you want to initialize the Macintosh Toolbox yourself, set the field `initializeTB` to `false`. The field `standalone` does not affect `initializeTB`.

For example, these lines set up SIOUX for an application that handles its own events, creates its own menus, and initializes the Toolbox:

```
SIOUXSettings.standalone = false;
SIOUXSettings.setupmenus = false;
SIOUXSettings.initializeTB = false;
```

path2fss

This function is similar to PBMakeFSSpec.

```
#include <path2fss.h>
OSErr __path2fss
   (const char * pathName, FSSpecPtr spec)
```

Table 29.2 path2fss

pathname	const char *	The path name
spec	FSSpecPtr	A file specification pointer

Remarks

This function is similar to PBMakeFSSpec with four major differences:

- Takes only a path name as input (as a C string) no parameter block.
- Only makes FSSpecs for files, not directories.
- Works on *any* HFS Mac (Mac 512KE, Mac Plus or later) under any system version that supports HFS.
- Deals correctly with MFS disks (correctly traps file names longer than 63 chars and returns bdNamErr).

Like PBMakeFSSpec, this function returns fnfErr if the specified file does not exist but the FSSpec is still valid for the purposes of creating a new file.

Errors are returned for invalid path names or path names that specify directories rather than files.

Macintosh only—this function may not be implemented on all Mac OS versions.

SIOUXHandleOneEvent

Handles an event for a SIOUX window.

```
#include <SIOUX.h>
Boolean SIOUXHandleOneEvent(EventRecord *event);
```

Table 29.3 SIOUXHandleOneEvent

event	EventRecord*	A pointer to a toolbox event

Remarks

Before you handle an event, call `SIOUXHandleOneEvent()` so SIOUX can update its windows when necessary. The argument `event` should be an event that `WaitNextEvent()` or `GetNextEvent()` returned. The function returns `true` if it handled the event and `false` if it didn't. If `event` is a `NULL` pointer, the function polls the event queue until it receives an event.

If it handles the event, `SIOUXHandleOneEvent()` returns `true`. Otherwise, `SIOUXHandleOneEvent()` returns `false`.

Macintosh only—this function may not be implemented on all Mac OS versions.

Listing 29.2 Example of SIOUXHandleOneEvent() Usage

```
void MyEventLoop(void)
{
   EventRecord event;
   RgnHandle cursorRgn;
   Boolean gotEvent, SIOUXDidEvent;

   cursorRgn = NewRgn();

   do {
      gotEvent = WaitNextEvent(everyEvent, &event,
                  MyGetSleep(), cursorRgn);

      /* Before handling the event on your own,
       * call SIOUXHandleOneEvent() to see whether
       * the event is for SIOUX.
       */
      if (gotEvent)
         SIOUXDidEvent = SIOUXHandleOneEvent(&event);

    if (!SIOUXDidEvent)
      DoEvent(&event);

   } while (!gDone)
}
```

SIOUXSetTitle

To set the title of the SIOUX output window.

```
include <SIOUX.h>
extern void SIOUXSetTitle(unsigned char title[256])
```

Table 29.4 SIOUXSetTitle

title	unsigned char []	A pascal string

Remarks

You must call the SIOUXSetTitle() function after an output to the SIOUX window. The function SIOUXSetTitle() does not return an error if the title is not set. A write to console is not performed until a new line is written, the stream is flushed or the end of the program occurs.

NOTE The argument for SIOUXSetTitle() is a pascal string, not a C style string.

There is no return value from SIOUXSetTitle()

Macintosh only—this function may not be implemented on all Mac OS versions.

Listing 29.3 Example of SIOUXSetTitle() Usage

```
#include <stdio.h>
#include <SIOUX.h>

int main(void)
{
   printf("Hello World\n");
   SIOUXSetTitle("\pMy Title");

   return 0;
}
```