WinAPI in ASM Intro

When a Windows application launches, it can create either a console window or a graphical window. In our project files, we've used the following option with the LINK command to specify a console-based application:

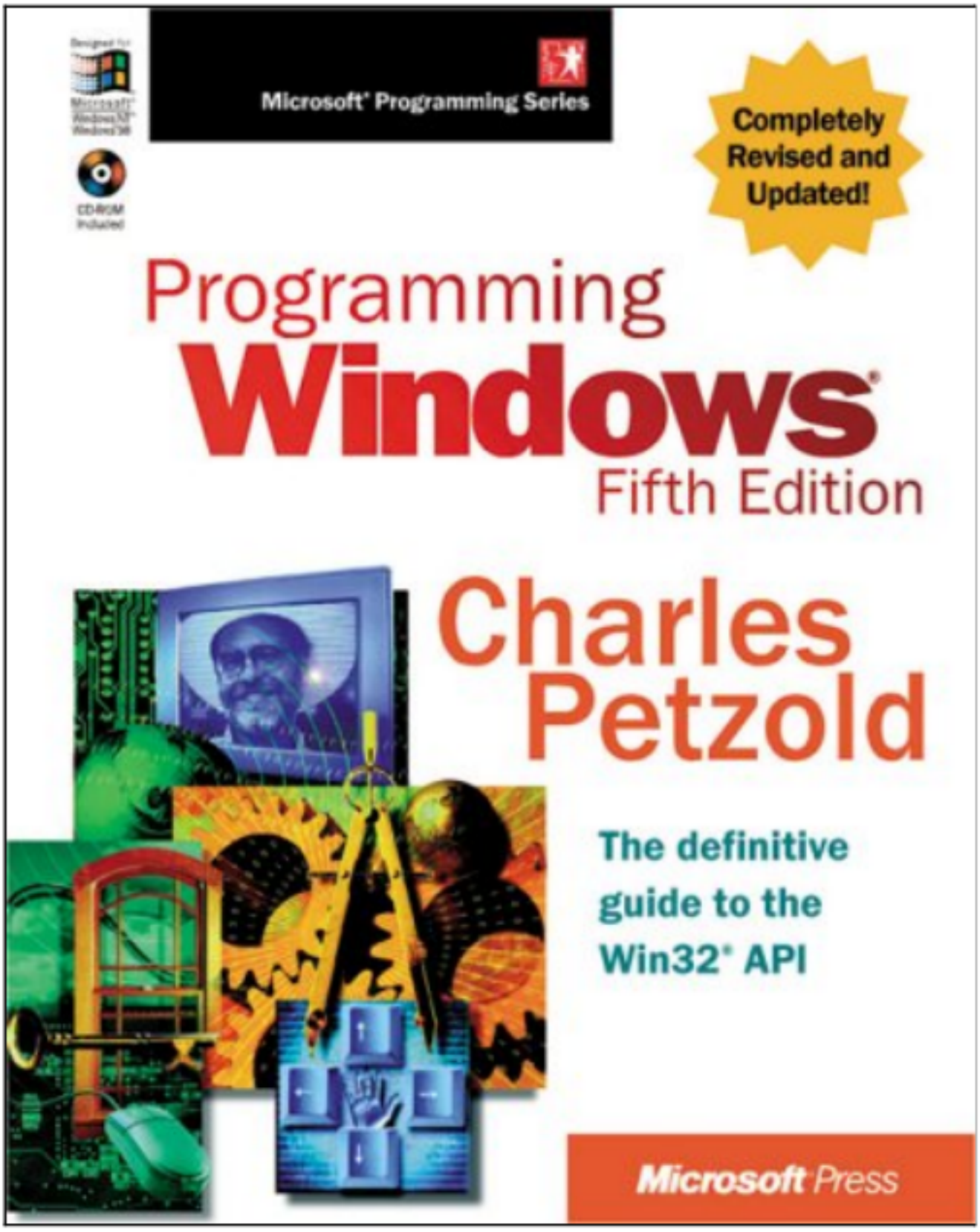
```
/SUBSYSTEM:CONSOLE
```

A console program resembles an MS-DOS window but with additional features, as we'll explore shortly.

It includes a single input buffer for queuing input records, which contain data about input events such as keyboard input, mouse clicks, and user actions like resizing the console window.

Additionally, it features one or more screen buffers, which are two-dimensional arrays containing character and color data that affect the appearance of text in the console window.

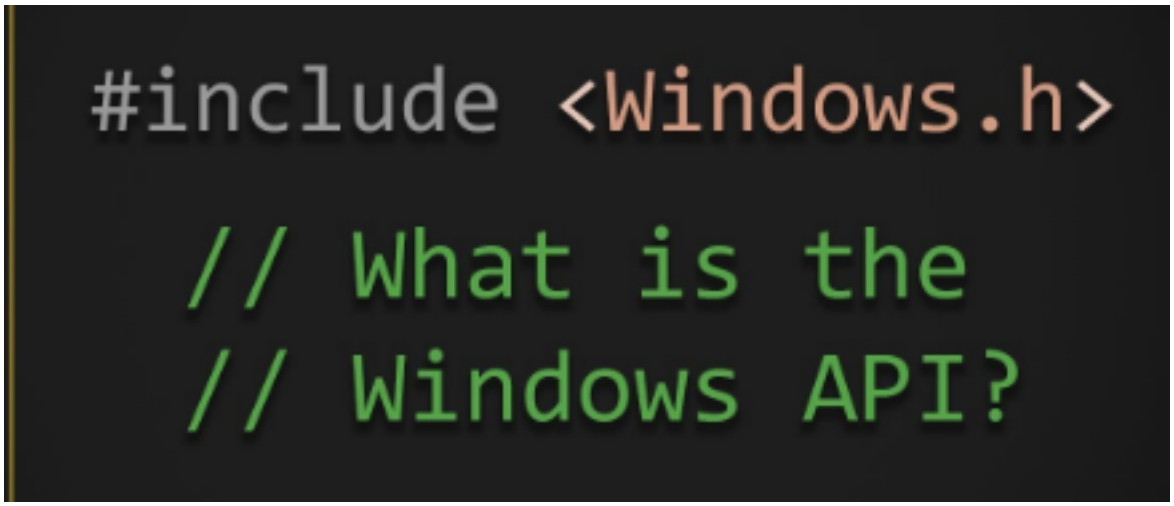
Win32 API Reference Information



Here is a summary of the key points:

• This section introduces a subset of Win32 API functions with simple examples, but does not cover every detail due to space constraints.
• The Microsoft MSDN website contains full documentation on the Win32 APIs. Make sure to filter for "Platform SDK" when searching.

• The sample programs include lists of function names in kernel32.lib and user32.lib libraries for reference.
• Win32 API functions often use named constants like TIME_ZONE_ID_UNKNOWN.

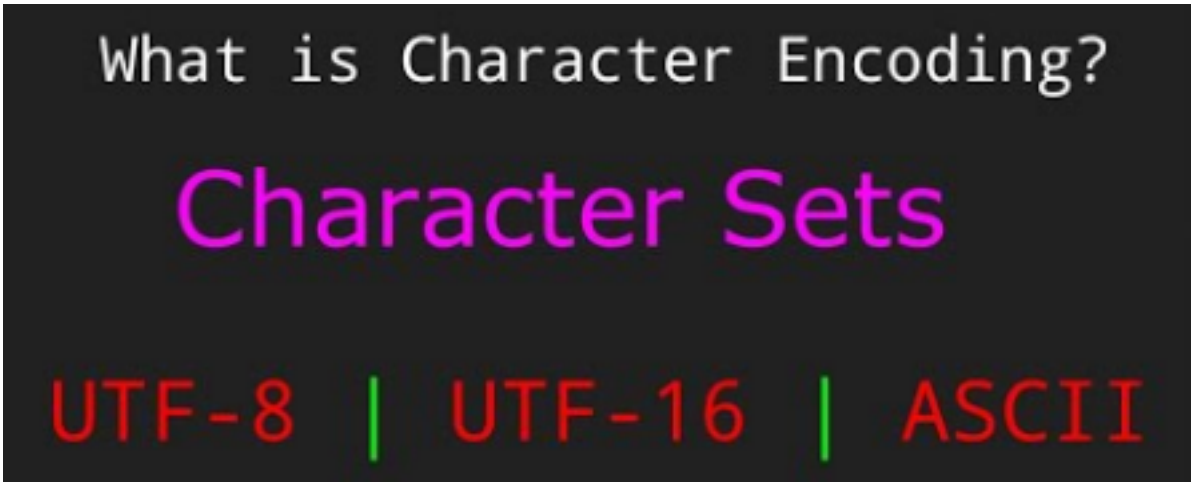


• Some constants are defined in SmallWin.inc, others can be found by referring to Windows header files like WinNT.h on the book's website.
• The header files define groups of related constants used by the Win32 functions.

• This overview provides a starting point on using Win32 APIs in assembly, but full details can be found in the Microsoft documentation and header files.
• The example code illustrates simple usage of some key functions.

Character Sets and Windows API Functions

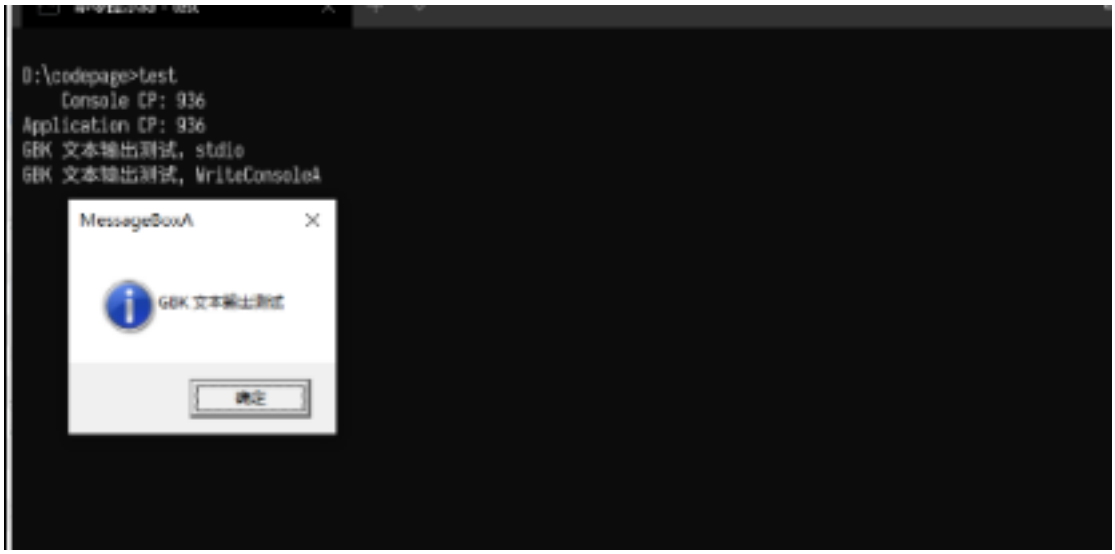
When calling functions in the Win32 API, two character sets are commonly used: the 8-bit ASCII/ANSI character set and the 16-bit Unicode set, which is available in recent Windows versions.



Win32 functions related to text come in two versions: one ending with 'A' (for 8-bit ANSI characters) and the other ending with 'W' (for wide character sets, including Unicode). For example, there are two versions of the WriteConsole function:

- **WriteConsoleA**
- **WriteConsoleW**

It's important to note that function names ending with 'W' are not supported in Windows 95 or 98.



In modern Windows versions, Unicode is the native character set. If you call a function like WriteConsoleA, the operating system performs character conversion from ANSI to Unicode and then calls WriteConsoleW.

In Microsoft's MSDN Library documentation, the trailing 'A' or 'W' is typically omitted from the function names.

In the program's include files provided with this book, function names like WriteConsoleA are redefined as follows:

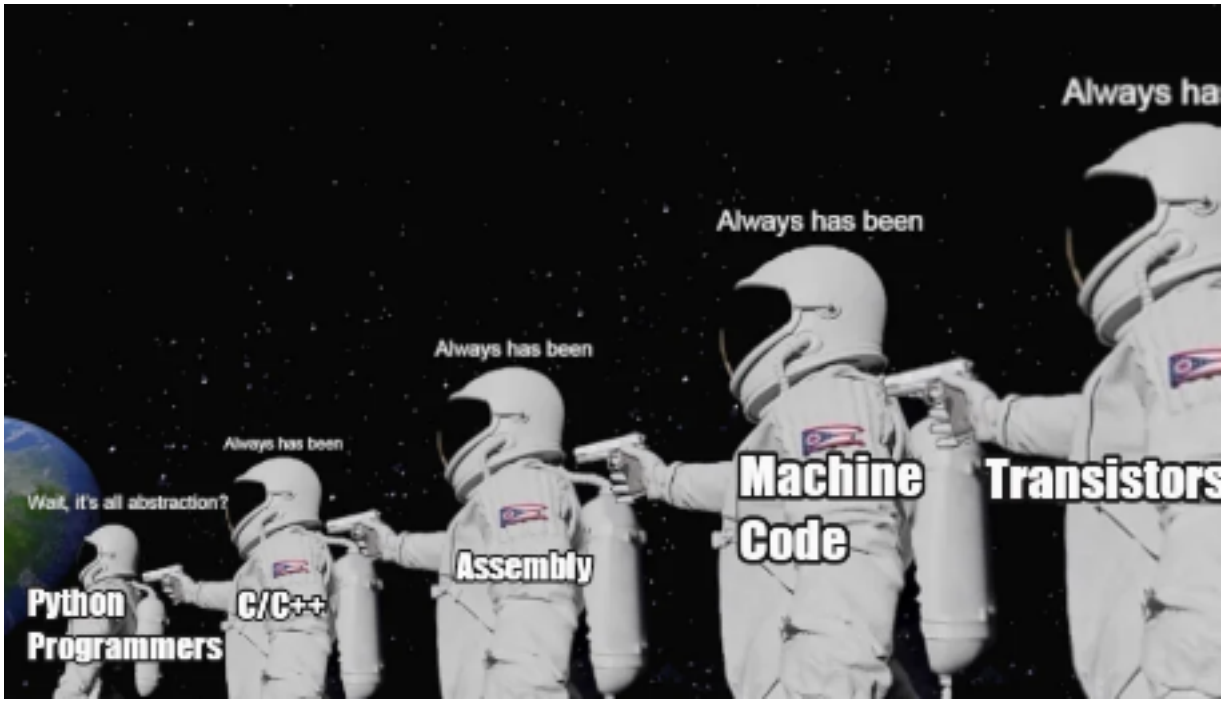
```
WriteConsole EQU <WriteConsoleA>
```

This definition allows you to call WriteConsole using the generic name.

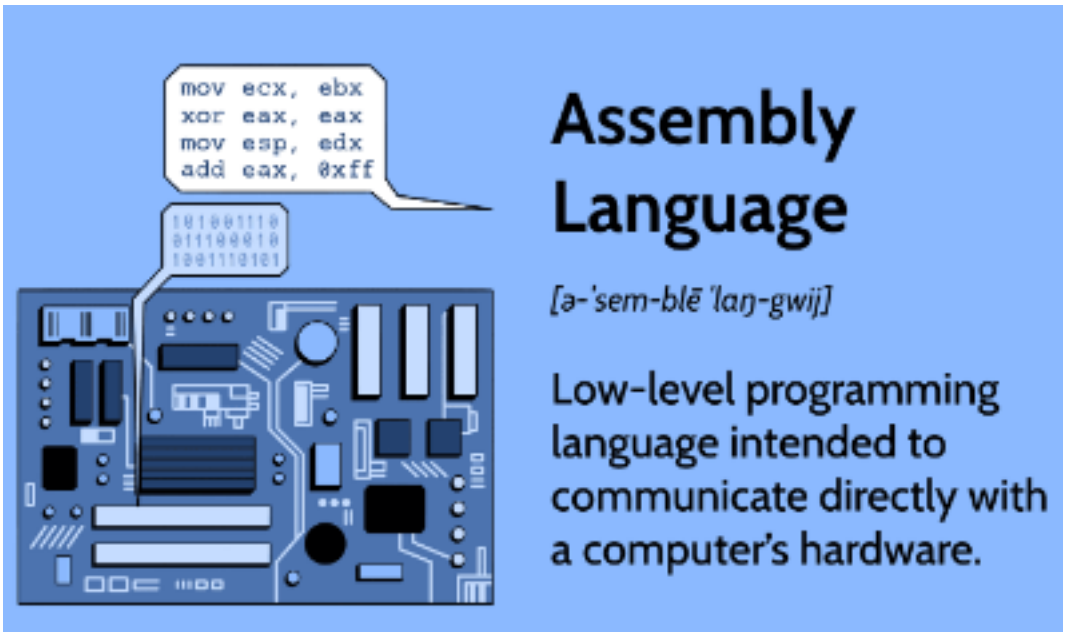
High-Level and Low-Level Console Access

There are two levels of access to the console, each offering a trade-off between simplicity and complete control:

High-Level Console Functions: These functions read a stream of characters from the console's input buffer and write character data to the console's screen buffer. Both input and output can be redirected to read from or write to text files.



Low-Level Console Functions: These functions provide detailed information about keyboard and mouse events, as well as user interactions with the console window (e.g., dragging, resizing). They also enable precise control over the window's size, position, and text colors.



This summary should provide you with a clear understanding of character sets and the distinctions between high-level and low-level console access in Windows API programming. If you have any further questions or need more information, please feel free to ask.

Windows Data Types

The MASM translations of the MS-Windows data types in Table 11-1 are as follows:

MS-Windows Type	MASM Type	Description
BOOL, BOOLEAN	DWORD	A boolean value (TRUE or FALSE)
BYTE	BYTE	An 8-bit unsigned integer
CHAR	BYTE	An 8-bit Windows ANSI character

In other words, the following MASM types are equivalent to the corresponding MS-Windows types:

```
01 DWORD = BOOL = BOOLEAN
02 BYTE = CHAR
```

It is important to note that the **HANDLE type in MS-Windows is also a DWORD**. This means that a HANDLE variable can be used to store a handle to any type of object, such as a window, a file, or a memory region.

Here is an example of how to declare and use a HANDLE variable in MASM:

```
handleVariable: DWORD

; Get a handle to the console window.
invoke GetConsoleWindow, handleVariable

; Use the handle to write a message to the console.
invoke WriteConsole, handleVariable, addr message, length message, bytesWritten, NULL

; Close the handle to the console window.
invoke CloseHandle, handleVariable
```

The SmallWin.inc include file contains constant definitions, text equates, and function prototypes for Win32 API programming.

It is automatically included in programs by Irvine32.inc. The file contains definitions for several Win32 data types, including the HANDLE type.

Here are some examples of how to use the SmallWin.inc include file:

```
; Get a handle to the standard input handle.
invoke GetStdHandle, STD_INPUT_HANDLE, handleVariable

; Get a handle to the standard output handle.
invoke GetStdHandle, STD_OUTPUT_HANDLE, handleVariable

; Get a handle to the standard error handle.
invoke GetStdHandle, STD_ERROR_HANDLE, handleVariable
```

The SmallWin.inc include file can be used to simplify the development of Win32 API programs in MASM.

Here is a clear and concise explanation of the MS-Windows data types listed in your notes:

BOOL, BOOLEAN: A boolean value, either TRUE or FALSE.

BYTE: An 8-bit unsigned integer, meaning that it can store values from 0 to 255.

CHAR: An 8-bit Windows ANSI character. ANSI characters are used in older Windows applications and are encoded using a variety of different character sets, depending on the language and region.

COORD: A structure that contains two WORD values, X and Y, which represent the coordinates of a point on the screen.

SYSTEMTIME: A structure that contains information about a date and time, including the year, month, day, hour, minute, second, and millisecond.

COLORREF: A 32-bit value used to represent a color.

DWORD: A 32-bit unsigned integer, meaning that it can store values from 0 to 4,294,967,295.

HANDLE: A handle is a reference to an object, such as a window, file, or memory region.

HFILE: A handle to a file opened by the OpenFile function.

INT: A 32-bit signed integer, meaning that it can store values from -2,147,483,648 to 2,147,483,647.

LONG: A 32-bit signed integer, the same as INT.

LPARAM: A message parameter used by window procedures and callback functions. LPARAM can be used to store any type of data, but it is typically used to store pointers to structures or other data structures.

LPCSTR: A pointer to a constant null-terminated string of 8-bit Windows (ANSI) characters.

LPCVOID: A pointer to a constant of any type.

LPSTR: A pointer to a null-terminated string of 8-bit Windows (ANSI) characters.

LPCTSTR: A pointer to a constant character string that is portable for Unicode and double-byte character sets. Unicode is a modern character encoding that can represent characters from all over the world. Double-byte character sets are used to represent characters in languages such as Chinese and Japanese.

LPTSTR: A pointer to a character string that is portable for Unicode and double-byte character sets.

LPVOID: A pointer to an unspecified type.

LRESULT: A 32-bit value returned from a window procedure or callback function.

SIZE_T: The maximum number of bytes to which a pointer can point.

UINT: A 32-bit unsigned integer, the same as DWORD.

WNDPROC: A pointer to a window procedure. A window procedure is a function that is responsible for handling messages sent to a window.

WORD: A 16-bit unsigned integer, meaning that it can store values from 0 to 65,535.

WPARAM: A 32-bit value passed as a parameter to a window procedure or callback function. WPARAM can be used to store any type of data, but it is typically used to store the message ID or other information about the message.

The SmallWin.inc include file contains structure definitions, data type definitions, and function prototypes for Win32 API programming. It is automatically included in MASM programs by the Irvine32.inc include file.

Structures Explained:

The **COORD structure** is used to store the coordinates of a point on the screen. It contains two WORD members, X and Y, which represent the horizontal and vertical coordinates of the point, respectively.

The **SYSTEMTIME structure** is used to store information about a date and time. It contains the following members:

- **wYear:** The year.
- **wMonth:** The month.
- **wDayOfWeek:** The day of the week.
- **wDay:** The day of the month.
- **wHour:** The hour.
- **wMinute:** The minute.
- **wSecond:** The second.
- **wMilliseconds:** The millisecond.

Console handles

Console handles are 32-bit unsigned integers that uniquely identify console devices, such as the keyboard, display, and printer. They are used by Win32 console functions to perform input and output operations.

The three standard console handles are:

STD_INPUT_HANDLE: The standard input handle is used to read keyboard input.

STD_OUTPUT_HANDLE: The standard output handle is used to write to the console display.

STD_ERROR_HANDLE: The standard error handle is used to write error messages to the console display. To get a handle to a console device, you can use the **GetStdHandle function.** This function takes a console handle type as a parameter and returns a handle to the corresponding console device.

Once you have a handle to a console device, you can use it to perform input and output operations. For example, to read a character from the keyboard, you can use the ReadConsole function. This function takes a console input handle and a buffer as parameters and reads a specified number of characters from the console input buffer into the buffer.

To write a character to the console display, you can use the WriteConsole function. This function takes a console output handle, a buffer, and a number of characters to write as parameters and writes the specified number of characters from the buffer to the console

display.

You can also use console handles to control the appearance and behavior of the console window. For example, to set the title of the console window, you can use the SetConsoleTitle function. This function takes a console window handle and a title string as parameters and sets the title of the console window to the specified string.

Console handles are an essential part of Win32 console programming. By understanding how to use console handles, you can develop powerful and efficient console-based applications.

The handles are:

AllocConsole
This function allocates a new console for the calling process. This is useful for applications that need to create their own console, such as console-based games or debugging tools.

CreateConsoleScreenBuffer
This function creates a new console screen buffer. A console screen buffer is a memory area that stores the text and color attributes for the console display.

ExitProcess
This function ends a process and all its threads. It is typically used to terminate an application when it is finished running or when an error occurs.

FillConsoleOutputAttribute
This function sets the text and background color attributes for a specified number of character cells. This can be used to change the appearance of text on the console display.

FillConsoleOutputCharacter
This function writes a character to the screen buffer a specified number of times. This can be used to fill a rectangular area of the console display with a single character.

FlushConsoleInputBuffer
This function flushes the console input buffer. The console input buffer is a memory area that stores keyboard input until it is read by an application. Flushing the console input buffer removes all

unread input from the buffer.

FreeConsole
This function detaches the calling process from its console. This is useful for applications that need to run without a console, such as services or background tasks.

GenerateConsoleCtrlEvent
This function sends a specified signal to a console process group that shares the console associated with the calling process. This can be used to notify other applications that the calling process is terminating or that an event has occurred.

GetConsoleCP
This function retrieves the input code page used by the console associated with the calling process. The input code page is a table that maps character codes to characters.

GetConsoleCursorInfo
This function retrieves information about the size and visibility of the cursor for the specified console screen buffer.

GetConsoleMode
This function retrieves the current input mode of a console input buffer or the current output mode of a console screen buffer. The input and output modes control the behavior of the console input and output, respectively.

GetConsoleOutputCP
This function retrieves the output code page used by the console associated with the calling process. The output code page is a table that maps characters to character codes.

GetConsoleScreenBufferInfo
This function retrieves information about the specified console screen buffer.

GetConsoleTitle
This function retrieves the title bar string for the current console window.

GetConsoleWindow
This function retrieves the window handle used by the console

associated with the calling process.

GetLargestConsoleWindowSize
This function retrieves the size of the largest possible console window.

GetNumberOfConsoleInputEvents
This function retrieves the number of unread input records in the console's input buffer.

GetNumberOfConsoleMouseButtons
This function retrieves the number of buttons on the mouse used by the current console.

GetStdHandle
This function retrieves a handle for the standard input, standard output, or standard error device. These handles are typically used by console applications to read keyboard input, write to the console display, and write error messages, respectively.

HandlerRoutine
This is an application-defined function that is used with the SetConsoleCtrlHandler function. The SetConsoleCtrlHandler function allows an application to specify a function to be called when the console receives certain signals, such as a close signal or a termination signal.

PeekConsoleInput
This function reads data from the specified console input buffer without removing it from the buffer. This can be used to check for keyboard input without actually reading it.

ReadConsole
This function reads character input from the console input buffer and removes it from the buffer. This is the most common way to read keyboard input in a console application.

ReadConsoleInput
This function reads data from a console input buffer and removes it from the buffer. This function is similar to the ReadConsole function, but it can also read mouse input and other types of input.

ReadConsoleOutput

This function reads character and color attribute data from a rectangular block of character cells in a console screen buffer. This can be used to read the text and appearance of a rectangular area of the console display.

ReadConsoleOutputAttribute

This function copies a specified number of foreground and background color attributes from consecutive cells of a console screen buffer. This can be used to read the color attributes of a rectangular area of the console display.

ReadConsoleOutputCharacter

This function copies a number of characters from consecutive cells of a console screen buffer. This can be used to read the text of a rectangular area of the console display.

ScrollConsoleScreenBuffer

This function moves a block of data in a screen buffer. This can be used to scroll the console display, or to move text or other data within the screen buffer.

SetConsoleActiveScreenBuffer

This function sets the specified screen buffer to be the currently displayed console screen buffer. This can be used to switch between different screen buffers, or to display a different screen buffer in a different console window.

SetConsoleCP

This function sets the input code page used by the console associated with the calling process. The input code page is a table that maps character codes to characters. This function can be used to change the language of the console input, or to support different character sets.

SetConsoleCtrlHandler

This function adds or removes an application-defined HandlerRoutine from the list of handler functions for the calling process. A HandlerRoutine is a function that is called when the console receives certain signals, such as a close signal or a termination signal. This function can be used to implement custom behavior when the console receives these signals.

SetConsoleCursorInfo

This function sets the size and visibility of the cursor for the

specified console screen buffer. This function can be used to change the appearance of the cursor, or to hide the cursor altogether.

SetConsoleCursorPosition
This function sets the cursor position in the specified console screen buffer. This function can be used to move the cursor to a specific location on the console display.

SetConsoleMode
This function sets the input mode of a console's input buffer or the output mode of a console screen buffer. The input and output modes control the behavior of the console input and output, respectively. This function can be used to change the behavior of the console keyboard, mouse, and other input devices, or to change the appearance of the console display.

SetConsoleOutputCP
This function sets the output code page used by the console associated with the calling process. The output code page is a table that maps characters to character codes. This function can be used to change the language of the console output, or to support different character sets.

SetConsoleScreenBufferSize
This function changes the size of the specified console screen buffer. This function can be used to increase or decrease the size of the console display, or to accommodate different screen sizes.

SetConsoleTextAttribute
This function sets the foreground (text) and background color attributes of characters written to the screen buffer. This function can be used to change the appearance of text on the console display.

SetConsoleTitle
This function sets the title bar string for the current console window. This can be used to change the title of the console window, or to identify the console window in a list of windows.

SetConsoleWindowInfo
This function sets the current size and position of a console screen buffer's window. This function can be used to resize or move the console window, or to fit the console window to a specific screen area.

SetStdHandle

This function sets the handle for the standard input, standard output, or standard error device. These handles are typically used by console applications to read keyboard input, write to the console display, and write error messages, respectively.

WriteConsole

This function writes a character string to a console screen buffer beginning at the current
cursor location. This is the most common way to write text to the console display.

WriteConsoleInput

This function writes data directly to the console input buffer. This function can be used to simulate keyboard input, or to send other types of input to the console.

WriteConsoleOutput

This function writes character and color attribute data to a specified rectangular block of character cells in a console screen buffer. This function can be used to write text and color attribute data to a specific area of the console display.

WriteConsoleOutputAttribute

This function copies a number of foreground and background color attributes to consecutive cells of a console screen buffer. This function can be used to change the color attribute of a specific area of the console display.

WriteConsoleOutputCharacter

This function copies a number of characters to consecutive cells of a console screen buffer. This function can be used to write text to a specific area of the console display.