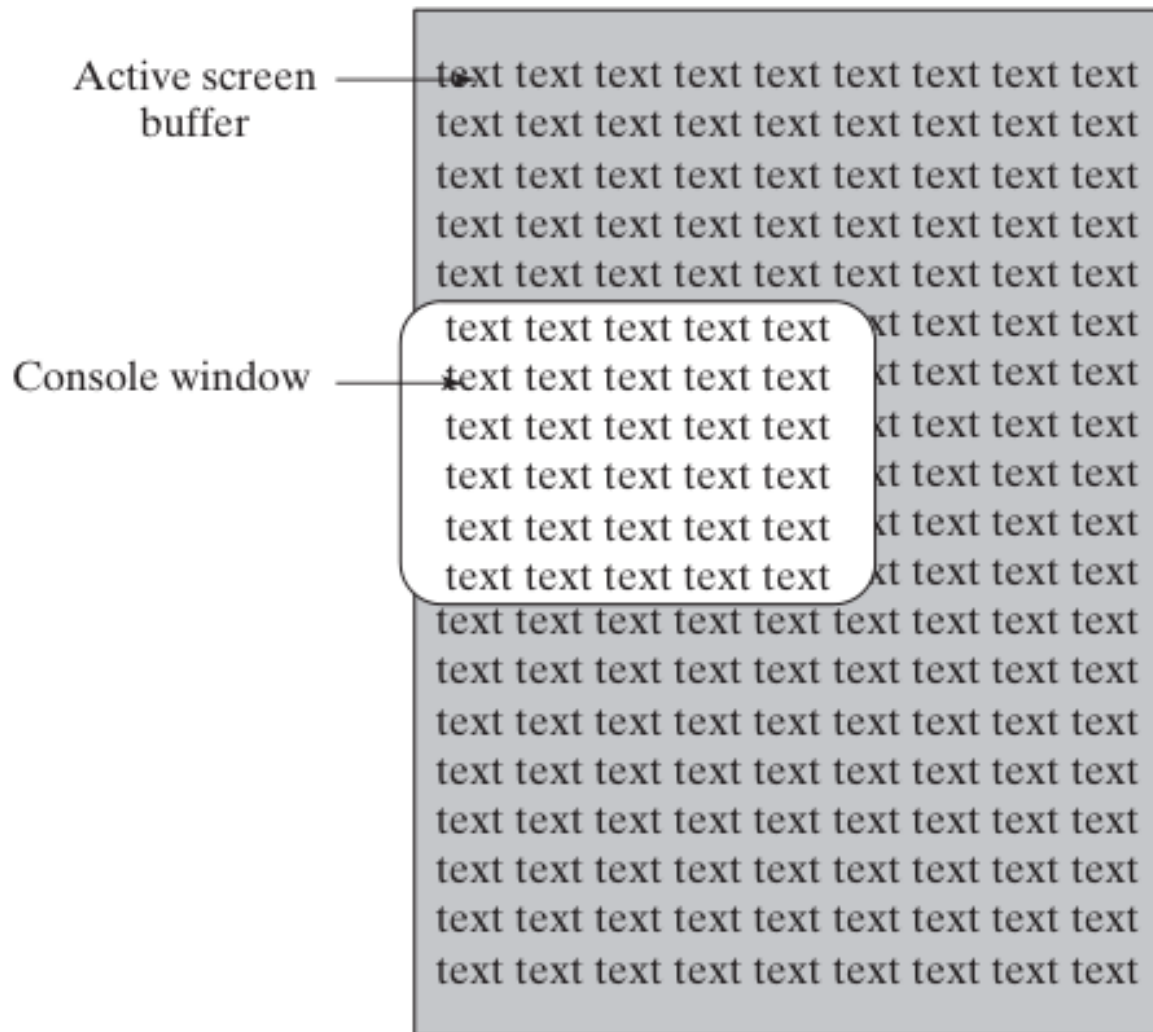


# Console Window Manipulation

I'll simplify the notes and provide commented code for each of the functions:

## Screen buffer and console window.



The image you sent shows a screen buffer and console window. The screen buffer is a memory area that stores the text and color attributes for the console display. The console window is the window that displays the console buffer.

To manipulate the screen buffer in assembly WinAPI, you can use the following functions:

**WriteConsoleOutput():** Writes character and color attribute data to a specified rectangular block of character cells in a console screen buffer.

**ReadConsoleOutput():** Reads character and color attribute data from a specified rectangular block of character cells in a console screen buffer.

**SetConsoleCursorPosition():** Sets the cursor position in the specified console screen buffer. The following code shows an example of how to use the WriteConsoleOutput function to write text to the screen buffer:

```
700 ; Get a handle to the console screen buffer.
701 mov eax, STD_OUTPUT_HANDLE
702 invoke GetStdHandle
703 mov ebx, eax
704
705 ; Set the cursor position.
706 mov ecx, 0 ; X coordinate
707 mov edx, 0 ; Y coordinate
708 invoke SetConsoleCursorPosition
709 mov esi, ebx
710
711 ; Write the text to the screen buffer.
712 mov edi, 0 ; X coordinate
713 mov edi, 0 ; Y coordinate
714 mov al, 'A'
715 invoke WriteConsoleOutput
716
717 ; Exit the program.
718 mov eax, 0
719 invoke ExitProcess
```

This code will write the character 'A' to the screen buffer at the top-left corner of the console window.

You can use the ReadConsoleOutput function to read text from the screen buffer. For example, the following code shows how to read a single character from the screen buffer:

```

722 ; Get a handle to the console screen buffer.
723 mov eax, STD_OUTPUT_HANDLE
724 invoke GetStdHandle
725 mov ebx, eax
726
727 ; Set the cursor position.
728 mov ecx, 0 ; X coordinate
729 mov edx, 0 ; Y coordinate
730 invoke SetConsoleCursorPosition
731 mov esi, ebx
732
733 ; Read a single character from the screen buffer.
734 mov edi, 0 ; X coordinate
735 mov edi, 0 ; Y coordinate
736 mov al, 1 ; Number of characters to read
737 invoke ReadConsoleOutput
738
739 ; Exit the program.
740 mov eax, 0
741 invoke ExitProcess

```

This code will read a single character from the screen buffer at the top-left corner of the console window.

You can use the `SetConsoleCursorPosition` function to set the cursor position in the screen buffer. For example, the following code shows how to set the cursor position to the middle of the console window:

```

745 ; Get a handle to the console screen buffer.
746 mov eax, STDOUT_HANDLE
747 invoke GetStdHandle
748 mov ebx, eax
749
750 ; Set the cursor position.
751 mov ecx, 40 ; X coordinate
752 mov edx, 25 ; Y coordinate
753 invoke SetConsoleCursorPosition
754 mov esi, ebx
755
756 ; Exit the program.
757 mov eax, 0
758 invoke ExitProcess

```

This code will set the cursor position to the middle of the console window.

### *SetConsoleTitle, GetConsoleScreenBufferInfo, and SetConsoleWindowInfo.*

```

665 ; SetConsoleTitle function to change the console window's title
666 .data
667 titleStr BYTE "New Console Title",0
668
669 .code
670 ; Invoke SetConsoleTitle with the specified title string
671 INVOKE SetConsoleTitle, ADDR titleStr

```

This code demonstrates how to change the console window's title using the SetConsoleTitle function.

```

675 ; GetConsoleScreenBufferInfo function to retrieve information about the console window
676 .data
677 consoleInfo CONSOLE_SCREEN_BUFFER_INFO <>
678 outHandle HANDLE ?
679
680 .code
681 ; Invoke GetConsoleScreenBufferInfo to retrieve information about the console window
682 INVOKE GetConsoleScreenBufferInfo, outHandle, ADDR consoleInfo

```

This code shows how to use the **GetConsoleScreenBufferInfo** function to obtain information about the console window, including screen buffer size, cursor position, and other details. The retrieved information is stored in the consoleInfo structure.

```

685 ; SetConsoleWindowInfo function to set the console window's size and position
686 .data
687 windowRect SMALL_RECT <0, 0, 79, 24> ; Example window rectangle
688
689 .code
690 ; Invoke SetConsoleWindowInfo to set the console window's size and position
691 INVOKE SetConsoleWindowInfo, outHandle, TRUE, ADDR windowRect

```

This code demonstrates how to use the **SetConsoleWindowInfo** function to set the size and position of the console window relative to the screen buffer. The windowRect structure defines the new window dimensions and position.

CONSOLE\_SCREEN\_BUFFER\_INFO structure.

Watch 1		
Name	Value	Type
consoleInfo	{dwSize={X=0x0078 Y=0x0032 } dwCursorPosition=	CONSOLE_SCREEN_BUFFER_INFO
dwSize	{X=0x0078 Y=0x0032 }	COORD
X	0x0078	unsigned short
Y	0x0032	unsigned short
dwCursorPosition	{X=0x0014 Y=0x0005 }	COORD
X	0x0014	unsigned short
Y	0x0005	unsigned short
wAttributes	0x0007	unsigned short
srWindow	{Left=0x0000 Top=0x0000 Right=0x004f ...}	SMALL_RECT
Left	0x0000	unsigned short
Top	0x0000	unsigned short
Right	0x004f	unsigned short
Bottom	0x0018	unsigned short
dwMaximumWindowSize	{X=0x0078 Y=0x0032 }	COORD
X	0x0078	unsigned short
Y	0x0032	unsigned short

I'll provide a simplified and commented version of the Scroll.asm

program:

```
767 INCLUDE Irvine32.inc
768
769 .data
770 message BYTE ": This line of text was written to the screen buffer",0dh,0ah
771 messageSize DWORD ($-message)
772 outHandle HANDLE 0 ; Standard output handle
773 bytesWritten DWORD ?
774 lineNum DWORD 0
775 windowRect SMALL_RECT <0,0,60,11> ; Left, top, right, bottom
776
777 .code
778 main PROC
779     ; Get the standard output handle
780     INVOKE GetStdHandle, STD_OUTPUT_HANDLE
781     mov outHandle, eax
782
783     .REPEAT
784         ; Display the line number
785         mov eax, lineNum
786         call WriteDec
787
788         ; Write the message to the console
789         INVOKE WriteConsole, outHandle, ADDR message, messageSize, ADDR bytesWritten, 0
790
791         ; Increment the line number
792         inc lineNum
793
794     .UNTIL lineNum > 50
795
796     ; Resize and reposition the console window
797     INVOKE SetConsoleWindowInfo, outHandle, TRUE, ADDR windowRect
798
799     ; Wait for a key press
800     call ReadChar
801
802     ; Clear the screen buffer
803     call Clrscr
804
805     ; Wait for a second key press
806     call ReadChar
807
808     ; Exit the program
809     INVOKE ExitProcess, 0
810
811 main ENDP
812
813 END main
```



This code simulates scrolling the console window by writing lines of text to the screen buffer and then resizing and repositioning the console window using `SetConsoleWindowInfo`. After running this program, press a key to trigger the scroll, clear the screen, and exit the program.

Another example:

```
819 INCLUDE Irvine32.inc
820
821 .data
822 consoleInfo CONSOLE_CURSOR_INFO <25, 1> ; Default cursor info
823 outHandle HANDLE 0
824 coord COORD <10, 10> ; New cursor position
825
826 .code
827 main PROC
828     ; Get the standard output handle
829     INVOKE GetStdHandle, STD_OUTPUT_HANDLE
830     mov outHandle, eax
831
832     ; Get the current cursor information
833     INVOKE GetConsoleCursorInfo, outHandle, ADDR consoleInfo
834
835     ; Display the current cursor size and visibility
836     mov eax, consoleInfo.dwSize
837     call WriteDec
838     call WriteString, ADDR " - Cursor Size, Visible: "
839     mov eax, consoleInfo.bVisible
840     call WriteDec
841     call Crlf
842
843     ; Set a new cursor size and visibility
844     mov consoleInfo.dwSize, 50
845     mov consoleInfo.bVisible, TRUE
846     INVOKE SetConsoleCursorInfo, outHandle, ADDR consoleInfo
847
```

```

848 ; Move the cursor to a new position
849 INVOKE SetConsoleCursorPosition, outHandle, ADDR coord
850
851 ; Display a message at the new cursor position
852 call WriteString, ADDR "New Cursor Position"
853
854 ; Wait for a key press
855 call ReadChar
856
857 ; Reset cursor info to the default values
858 mov consoleInfo.dwSize, 25
859 mov consoleInfo.bVisible, TRUE
860 INVOKE SetConsoleCursorInfo, outHandle, ADDR consoleInfo
861
862 ; Move the cursor back to the original position
863 mov coord.X, 0
864 mov coord.Y, 0
865 INVOKE SetConsoleCursorPosition, outHandle, ADDR coord
866
867 ; Display a message at the original cursor position
868 call WriteString, ADDR "Original Cursor Position"
869
870 ; Wait for a key press to exit
871 call ReadChar
872
873 INVOKE ExitProcess, 0
874 main ENDP
875
876 END main

```

This program demonstrates the usage of cursor control functions. It first retrieves the current cursor info, changes the cursor size and visibility, and moves the cursor to a new position. After displaying a message, it resets the cursor to its original state and waits for a key press before exiting.