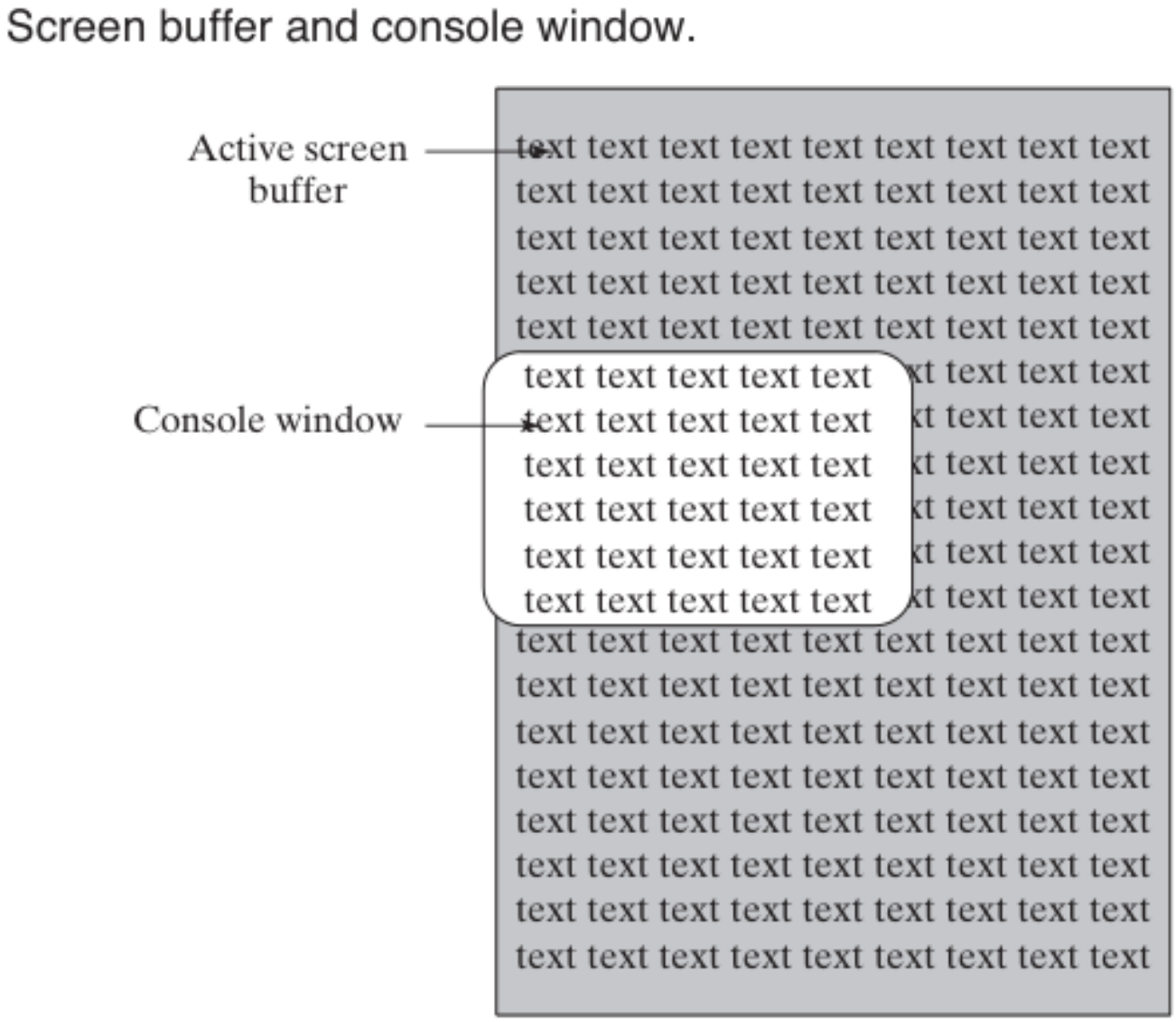
Console Window Manipulation

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



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:

```
; Get a handle to the console screen buffer.
mov eax, STD_OUTPUT_HANDLE
invoke GetStdHandle
mov ebx, eax

; Set the cursor position.
mov ecx, 0 ; X coordinate
mov edx, 0 ; Y coordinate
invoke SetConsoleCursorPosition
mov esi, ebx

; Write the text to the screen buffer.
mov edi, 0 ; X coordinate
mov edi, 0 ; Y coordinate
mov al, 'A'
invoke WriteConsoleOutput

; Exit the program.
mov eax, 0
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:

```
; Get a handle to the console screen buffer.
mov eax, STD_OUTPUT_HANDLE
invoke GetStdHandle
mov ebx, eax

; Set the cursor position.
mov ecx, 0 ; X coordinate
mov edx, 0 ; Y coordinate
invoke SetConsoleCursorPosition
mov esi, ebx

; Read a single character from the screen buffer.
mov edi, 0 ; X coordinate
mov edi, 0 ; Y coordinate
mov al, 1 ; Number of characters to read
invoke ReadConsoleOutput

; Exit the program.
mov eax, 0
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:

```
; Get a handle to the console screen buffer.
mov eax, STD_OUTPUT_HANDLE
invoke GetStdHandle
mov ebx, eax

; Set the cursor position.
mov ecx, 40 ; X coordinate
mov edx, 25 ; Y coordinate
invoke SetConsoleCursorPosition
mov esi, ebx

; Exit the program.
mov eax, 0
invoke ExitProcess
```

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

SetConsoleTitle, GetConsoleScreenBufferInfo, and SetConsoleWindowInfo.

```
; SetConsoleTitle function to change the console window's title
.data
titleStr BYTE "New Console Title",0

.code
; Invoke SetConsoleTitle with the specified title string
INVOKE SetConsoleTitle, ADDR titleStr
```

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

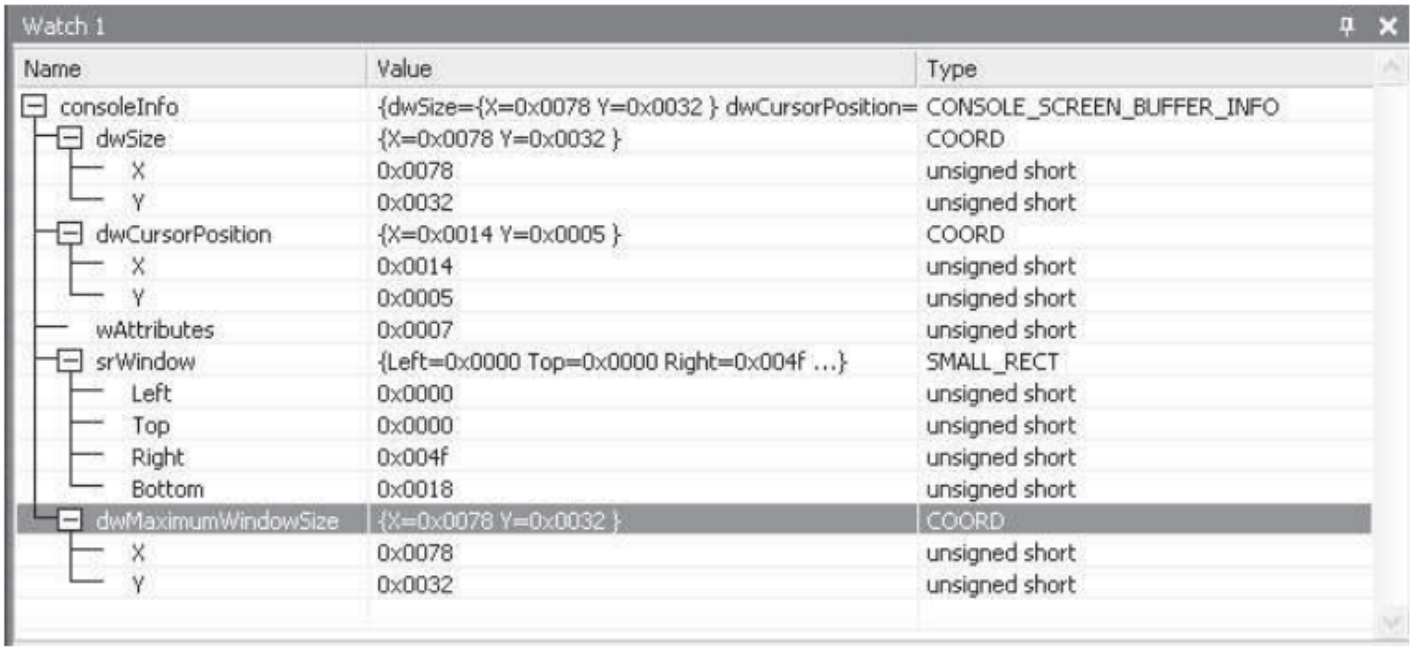
```
675 ; GetConsoleScreenBufferInfo function to retrieve information about the console window
676 .data
677 consoleInfo CONSOLE_SCREEN_BUFFER_INFO <>
678 outHandle HANDLE ?
679
680 .code
681 ; Invoke GetConsoleScreenBufferInfo to retrieve information about the console window
682 INVOKE GetConsoleScreenBufferInfo,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



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

program:

```
INCLUDE Irvine32.inc

.data
message BYTE ": This line of text was written to the screen buffer",0dh,0ah
messageSize DWORD ($-message)
outHandle HANDLE 0 ; Standard output handle
bytesWritten DWORD ?
lineNum DWORD 0
windowRect SMALL_RECT <0,0,60,11> ; Left, top, right, bottom

.code
main PROC
    ; Get the standard output handle
    INVOKE GetStdHandle, STD_OUTPUT_HANDLE
    mov outHandle, eax

    .REPEAT
        ; Display the line number
        mov eax, lineNum
        call WriteDec

        ; Write the message to the console
        INVOKE WriteConsole, outHandle, ADDR message, messageSize, ADDR bytesWritten, 0

        ; Increment the line number
        inc lineNum

    .UNTIL lineNum > 50

    ; Resize and reposition the console window
    INVOKE SetConsoleWindowInfo, outHandle, TRUE, ADDR windowRect
```

```
    ; Wait for a key press
    call ReadChar

    ; Clear the screen buffer
    call Clrscr

    ; Wait for a second key press
    call ReadChar

    ; Exit the program
    INVOKE ExitProcess, 0

main ENDP

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:

```
INCLUDE Irvine32.inc

.data
consoleInfo CONSOLE_CURSOR_INFO <25, 1> ; Default cursor info
outHandle HANDLE 0
coord COORD <10, 10> ; New cursor position

.code
main PROC
    ; Get the standard output handle
    INVOKE GetStdHandle, STD_OUTPUT_HANDLE
    mov outHandle, eax

    ; Get the current cursor information
    INVOKE GetConsoleCursorInfo, outHandle, ADDR consoleInfo

    ; Display the current cursor size and visibility
    mov eax, consoleInfo.dwSize
    call WriteDec
    call WriteString, ADDR " - Cursor Size, Visible: "
    mov eax, consoleInfo.bVisible
    call WriteDec
    call Crlf

    ; Set a new cursor size and visibility
    mov consoleInfo.dwSize, 50
    mov consoleInfo.bVisible, TRUE
    INVOKE SetConsoleCursorInfo, outHandle, ADDR consoleInfo

```

```
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.