High-Level Console Modes

07/12/2018 • 6 minutes to read •

The behavior of the high-level console functions is affected by the console input and output modes. All of the following console input modes are enabled for a console's input buffer when a console is created:

- Line input mode
- Processed input mode
- Echo input mode

Both of the following console output modes are enabled for a console screen buffer when it is created:

- Processed output mode
- Wrapping at EOL output mode

All three input modes, along with processed output mode, are designed to work together. It is best to either enable or disable all of these modes as a group. When all are enabled, the application is said to be in "cooked" mode, which means that most of the processing is handled for the application. When all are disabled, the application is in "raw" mode, which means that input is unfiltered and any processing is left to the application.

An application can use the **GetConsoleMode** function to determine the current mode of a console's input buffer or screen buffer. You can enable or disable any of these modes by using the following values in the **SetConsoleMode** function. Note that setting the output mode of one screen buffer does not affect the output mode of other screen buffers.

If the *hConsoleHandle* parameter is an input handle, the mode can be one or more of the following values. When a console is created, all input modes except **ENABLE_WINDOW_INPUT** are enabled by default.

Value	Meaning
ENABLE_ECHO_INPUT 0x0004	Characters read by the **ReadFile** or **ReadConsole** function are written to the active screen buffer as they are read. This mode can be used only if the **ENABLE_LINE_INPUT** mode is also enabled.
ENABLE_INSERT_MODE 0x0020	When enabled, text entered in a console window will be inserted at the current cursor location and all text following that location will not be overwritten. When disabled, all following text will be overwritten.
ENABLE_LINE_INPUT 0x0002	The **ReadFile** or **ReadConsole** function returns only when a carriage return character is read. If this mode is disabled, the functions return when one or more characters are available.

Value	Meaning
ENABLE_MOUSE_INPUT 0x0010	If the mouse pointer is within the borders of the console window and the window has the keyboard focus, mouse events generated by mouse movement and button presses are placed in the input buffer. These events are discarded by **ReadFile** or **ReadConsole**, even when this mode is enabled.
ENABLE_PROCESSED_INPUT 0x0001	CTRL+C is processed by the system and is not placed in the input buffer. If the input buffer is being read by **ReadFile** or **ReadConsole**, other control keys are processed by the system and are not returned in the **ReadFile** or **ReadConsole** buffer. If the **ENABLE_LINE_INPUT** mode is also enabled, backspace, carriage return, and line feed characters are handled by the system.
ENABLE_QUICK_EDIT_MODE 0x0040	This flag enables the user to use the mouse to select and edit text. To enable this mode, use `ENABLE_QUICK_EDIT_MODE | ENABLE_EXTENDED_FLAGS`. To disable this mode, use **ENABLE_EXTENDED_FLAGS** without this flag.
ENABLE_WINDOW_INPUT 0x0008	User interactions that change the size of the console screen buffer are reported in the console's input buffer. Information about these events can be read from the input buffer by applications using the **ReadConsoleInput** function, but not by those using **ReadFile** or **ReadConsole**.
ENABLE_VIRTUAL_TERMINAL_INPUT 0x0200	Setting this flag directs the Virtual Terminal processing engine to convert user input received by the console window into **Console Virtual Terminal Sequences** that can be retrieved by a supporting application through **WriteFile** or **WriteConsole** functions. The typical usage of this flag is intended in conjunction with ENABLE_VIRTUAL_TERMINAL_PROCESSING on the output handle to connect to an application that communicates exclusively via virtual terminal sequences.

If the *hConsoleHandle* parameter is a screen buffer handle, the mode can be one or more of the following values. When a screen buffer is created, both output modes are enabled by default.

Value	Meaning
ENABLE_PROCESSED_OUTPUT 0x0001	Characters written by the **WriteFile** or **WriteConsole** function or echoed by the **ReadFile** or **ReadConsole** function are parsed for ASCII control sequences, and the correct action is performed. Backspace, tab, bell, carriage return, and line feed characters are processed.

Value	Meaning
ENABLE_WRAP_AT_EOL_OUTPUT 0x0002	When writing with **WriteFile** or **WriteConsole** or echoing with **ReadFile** or **ReadConsole**, the cursor moves to the beginning of the next row when it reaches the end of the current row. This causes the rows displayed in the console window to scroll up automatically when the cursor advances beyond the last row in the window. It also causes the contents of the console screen buffer to scroll up (../discarding the top row of the console screen buffer) when the cursor advances beyond the last row in the console screen buffer. If this mode is disabled, the last character in the row is overwritten with any subsequent characters.
ENABLE_VIRTUAL_TERMINAL_PROCESSING 0x0004	When writing with **WriteFile** or **WriteConsole**, characters are parsed for VT100 and similar control character sequences that control cursor movement, color/font mode, and other operations that can also be performed via the existing Console APIs. For more information, see **Console Virtual Terminal Sequences**.
DISABLE_NEWLINE_AUTO_RETURN 0x0008	When writing with **WriteFile** or **WriteConsole**, this adds an additional state to end-of-line wrapping that can delay the cursor move and buffer scroll operations. Normally when **ENABLE_WRAP_AT_EOL_OUTPUT** is set and text reaches the end of the line, the cursor will immediately move to the next line and the contents of the buffer will scroll up by one line. In contrast with this flag set, the scroll operation and cursor move is delayed until the next character arrives. The written character will be printed in the final position on the line and the cursor will remain above this character as if **ENABLE_WRAP_AT_EOL_OUTPUT** was off, but the next printable character will be printed as if **ENABLE_WRAP_AT_EOL_OUTPUT** is on. No overwrite will occur. Specifically, the cursor quickly advances down to the following line, a scroll is performed if necessary, the character is printed, and the cursor advances one more position. The typical usage of this flag is intended in conjunction with setting **ENABLE_VIRTUAL_TERMINAL_PROCESSING** to better emulate a terminal emulator where writing the final character on the screen (../in the bottom right corner) without triggering an immediate scroll is the desired behavior.

Value	Meaning
ENABLE_LVB_GRID_WORLDWIDE 0x0010	The APIs for writing character attributes including **WriteConsoleOutput** and **WriteConsoleOutputAttribute** allow the usage of flags from **character attributes** to adjust the color of the foreground and background of text. Additionally, a range of DBCS flags was specified with the COMMON_LVB prefix. Historically, these flags only functioned in DBCS code pages for Chinese, Japanese, and Korean languages. With exception of the leading byte and trailing byte flags, the remaining flags describing line drawing and reverse video (../swap foreground and background colors) can be useful for other languages to emphasize portions of output. Setting this console mode flag will allow these attributes to be used in every code page on every language. It is off by default to maintain compatibility with known applications that have historically taken advantage of the console ignoring these flags on non-CJK machines to store bits in these fields for their own purposes or by accident. Note that using the ENABLE_VIRTUAL_TERMINAL_PROCESSING mode can result in LVB grid and reverse video flags being set while this flag is still off if the attached application requests underlining or inverse video via **Console Virtual Terminal Sequences**.