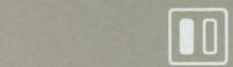


# Concurrent CP/M™ with Windows

Operating System

# Technical Note

for the IBM® Personal Computer
and Personal Computer XT


# Concurrent CP/M™ with Windows

Operating System

# Technical Note

for the IBM® Personal Computer
and Personal Computer XT

The Concurrent CP/M with Windows Operating System Technical Note for the IBM Personal Computer and Personal Computer XT was prepared using the Digital Research TEX™ Text Formatter and printed in the United States of America.

**********
* First Edition: February 1984 *
**********

# Table of Contents



# Tables and Figures

**Tables**



**Figures**

# Section 1
# Window Management for Programmers

This section describes how you can use the window management primitives of the Digital Research® retail version of Concurrent CP/M™ with Windows for the IBM® Personal Computer and Personal Computer XT. These window management primitives are implemented in the hardware-specific portion of the operating system (the XIOS). Any program that uses the window management primitives (or any other XIOS function) will probably not be portable to another Concurrent CP/M 3.1 system.

## 1.1 Console Output

There are four methods of handling console output in Concurrent CP/M. As you move from one method to another, the resulting console output has a higher degree of performance and a lower degree of portability. These four methods and their relative advantages and disadvantages are as follows:

### Method 1

This method uses the traditional BDOS system calls for console output with TTY-style output (that is, there is no cursor positioning or other escape sequences). This method works on any version of CP/M®, works inside windows if they are provided, and works on almost any kind of output device. However, this method is not useful for interactive applications such as word processors or spreadsheets. It is also not efficient for clearing the screen or positioning text at random positions on the screen.

### Method 2

This method uses the traditional BDOS system calls for console output with the terminal-oriented escape sequences that are documented for the terminal. This method works on most versions of CP/M, and works inside windows. It also works on almost any kind of terminal or memory map as long as the XIOS interprets the escape sequences appropriately. However, this method is not efficient for moving blocks of text quickly on the console or changing the attributes of blocks of text. It cannot use the full capabilities of the IBM PC character map.

Method 3

This method uses the XIOS backdoor entry points that allow you to write directly to the virtual console buffer and still work in windows. This method works inside windows, is extremely efficient, and exploits the full capabilities of the IBM PC character map. However, this method is nonportable, requires complex programming interface to the XIOS, and requires the ability to link with an assembly language routine.

Method 4

This method bypasses the BDOS and the XIOS by going directly to the physical console buffer (for either the monochrome monitor or the color monitor) as if it were the only process running in the system. This method is extremely efficient and exploits the full capabilities of the IBM PC character map. Also the code will run without change on CP/M-86® for the IBM PC. However, it is nonportable and does not work in windows. That is, applications can only be run full-screen and in the front screen layer.

Given these four alternatives for handling console output, you should choose the method best suited to the application. The techniques used for Methods 1, 2, and 4 are self-explanatory. The technique for Method 3 is described in Section 1.2. A sample program, SAMPLE.C, that uses Method 3 is included in the distribution diskette.

In addition to doing console output in a way that works with windows, you might want to directly control the placement, sizing, scrolling, tracking, and layering (what is on top) of the windows onto Concurrent CP/M's virtual consoles. This direct interaction with the window management primitives can be very useful. However, it is very XIOS-dependent and therefore nonportable. The functions for controlling windows directly are described in Section 1.3.

## 1.2 Technique for Method 3 Type of Console Output

Method 3 type of console output allows an application program to do efficient screen output and yet work within the confines of the Concurrent CP/M windows. In Concurrent CP/M each process that does console output must own a virtual console. The virtual console is actually an area of memory the same size and format as the physical console. The virtual console buffer saves the output that is done by the process while another process is using all or part of the physical console.

The XIOS in Concurrent CP/M ensures that the virtual console images are updated and that the physical console displays the appropriate portions of each virtual console. In the first version of Concurrent CP/M for the IBM PC, simple window management only allowed full-screen (24x80) windows to be displayed on the physical console. As the user switched screens, the virtual console that was brought to the front completely covered the physical console. However, in this latest version of Concurrent CP/M, the XIOS contains more sophisticated window management functions. Now, windowed portions of each virtual console may all be shown on the physical console at the same time.

The fact that different portions of a virtual console can be displayed on the physical console in almost any location makes it almost impossible for application programmers to go directly to the physical console and still work in windows. To provide a mechanism for efficient screen updates, a scheme has been devised that allows a program to update its virtual console buffer and then have the XIOS handle the windowing of the appropriate portion of the virtual console to the physical console.

The general steps you take for the Method 3 type of console output are as follows:

1. Make the BDOS function call to find your default virtual console number:

   ```
   vc_number = __BDOS( 0x99, 0x0);
   ```

2. Make a special XIOS call to find the address of the virtual console screen buffer that corresponds to your default virtual console:

   ```
   WM_PK(1, vc_number, VS_VC_SEGMENT, &vc_segment);
   ```

3. Make a special XIOS call to guarantee that what is currently on the physical console is synchronized with your virtual console screen buffer. This might become out of synch if another program has been sending direct output to your physical console, or if the virtual console window is full screen on top and positioned at 1,1 on the physical console:

   ```
   wm_sync(vc_number);
   ```

4. You are now ready to update your virtual console buffer directly. To guarantee correct operation, first make a special XIOS call that takes ownership of an internal semaphore that protects each virtual console screen buffer from two processes updating it at the same time:

   ```
   WM_MXQ(0);
   ```

5. You can now use the full power of the 8086 instruction set to update your virtual console screen buffer. After updating the buffer, immediately make a special XIOS call that frees the internal virtual console semaphore. No other XIOS window management functions that involve your virtual console can occur while you have the semaphore that protects your virtual console:

   ```
   WM_MXQ(1);
   ```

6. Finally, whenever you feel the time is right, make a special XIOS call that updates the physical console with the current contents of the virtual console:

   ```
   IO_CALL(&ax, &bx, &cx, &dx);
   ```

Steps 4, 5, and 6 can be repeated as often as necessary. Any time you mix traditional BDOS system calls for console output with direct updates of your virtual screen buffer, you must again perform step 3. The reason for this is that a special case of full screen on top on the monochrome display is made where the XIOS does not bother to keep the virtual console and the physical console synchronized. This allows for programs that go directly to the monochrome display to work as long as they are only run full screen and on top.

## 1.3 XIOS Entry Points for Window Management

This section contains the XIOS entry points that are specifically used for window control. It describes nine routines in terms of registers at entry and exit. These XIOS routines are "back door" in the sense that they cannot be called using the standard Function 50 XIOS calling convention. These routines are called using a far call to the XIOS entry point using the standard XIOS segment register conventions, specifically: DS = SYSDAT and ES = UDA.

At entry, each routine is called with a function code in the AL register, and various parameters in BX, CX, and DX. The routines and their decimal function codes are summarized in Table 1-1.

**Table 1-1. XIOS Window Functions**

| Routine | Code | Function |
|---|---|---|
| io_pointer | 16 | returns pointers to window data |
| io_key | 17 | returns a character to manager |
| io_nstatline | 18 | new status line call with attributes |
| io_im_here | 19 | sets the manager process state |
| io_new_window | 20 | sets a new console window |
| io_cursor_view | 21 | sets cursor track mode and viewpoint |
| io_wrap_column | 22 | sets the column for auto wrap-around |
| io_full | 23 | toggles VC from full to not full |
| io_display | 24 | sets which console the VC will be on |

The parameters for the routines are described in the following pages. The virtual console structure is described in Table 1-2, and the window data block is described in Table 1-3.

| Function Code 16 | io_pointer |
|---|---|
| Returns pointers to window-relevant information. | |

```
Entry Parameters:
     Register AL:  16
               DL = virtual console number
                  = 0FFH

 Returned Values:
     Register AX = vc structure pointer if
                   DL = virtual console number
                  = window data block pointer if
                   DL = 0FFH
```

**Table 1-2. Virtual Console Structure Definition**

| (Returned by io_pointer when DL = vc number) | | | |
|---|---|---|---|
| vs_cursor | word ptr | 00 | cursor row,col position in vc image |
| vs_top_left | word ptr | 02 | inside top,left corner of vc window |
| vs_bot_right | word ptr | 04 | inside bottom,right corner of window |
| vs_old_t_l | word ptr | 06 | saves the previous value of top_left |
| vs_old_b_r | word ptr | 08 | saves the previous value of bot_right |
| vs_crt_size | word ptr | 10 | total number of rows,cols in vc image |
| vs_win_size | word ptr | 12 | used to remember the size of the user's window by the WMENU window manager, this isn't updated by the XIOS |
| vs_view_point | word ptr | 14 | top_left image corner to top_left win |
| vs_rows | word ptr | 16 | number of rows in current vc window |
| vs_cols | word ptr | 18 | number of columns in current window |
| vs_correct | word ptr | 20 | window tracking correction factor |
| vs_vc_seg | word ptr | 22 | segment address of vc console image |
| vs_crt_seg | word ptr | 24 | segment address of crt memory |
| vs_list_ptr | word ptr | 26 | points to the start of row update list |
| vs_attrib | byte ptr | 28 | current vc character attribute |
| vs_mode | byte ptr | 29 | cursor on/off and wrap on/off |

**Table 1-2. (continued)**

| | | |
|---|---|---|
| vs_cur_track | byte ptr 30 | window fixed or tracking scroll |
| vs_width | byte ptr 31 | column at which to wrap around |
| vs_number | byte ptr 32 | copy of the virtual console number |
| vs_bit | byte ptr 33 | vc num = bit pos |
| vs_save_cursor | word ptr 34 | for ESC code save/restore cursor |
| vs_vector | word ptr 36 | conout state machine vector |
| vs_xlat | word ptr 38 | mono/color xlat table |
| vs_qpb | word ptr 40 | reserved |
| vs_true_view | word ptr 42 | corrected view point |
| vs_cur_type | word ptr 44 | mono or color |
| | . | |
| | . | |
| | . | |
| vs_mxsemaphore | byte ptr 52 | internal XIOS semaphore |

**Table 1-3. Window Data Block Definition**

| (Returned by io_pointer when DL = 0FFH) | | |
|---|---|---|
| im_here | byte ptr 0 | manager process state variable<br>0 => manager not resident<br>1 => manager resident but not active<br>2 => manager resident and active |
| nvc | byte ptr 1 | number of virtual consoles |
| priority | byte ptr 2 | a list of vc numbers (nvc bytes long) from the back window to front window |

| Function Code 17 | io_key |
|---|---|
| Return character to window manager. | |

```
Entry Parameters:
     Register AL: 17
                CL = 0FFH (input/status)
                   = 0FEH (status only)
                   < 0FFH (wait for input)

 Returned Values:
     Register AL = char if char ready (input/status)
                 = 0 if no char is ready (input/status)
                 = 0FFH if char ready (status only)
                 = 0 if no char is ready (status only)
                 = char when ready (wait for input)
                AH = key type if AL = char
                     (0=regular, 0FFH = special)
```

Window manager will go to sleep on this call if CL < 0FEH.

| Function Code 18 | io_nstatline |
|---|---|
| New status line call with attributes. | |

```
Entry Parameters:
     Register AL: 18
                  CX:
                  DX:
 Returned Values:
                  None
```

CX and DX are as described in the Concurrent CP/M Operating System System Guide.

This function works just like the normal io_statline call as described in the Concurrent CP/M System Guide, except the string pointed to contains not 80 characters, but 80 character/attribute pairs. The attributes are as defined in the Technical Reference Manual for the IBM PC.

| Function Code 19 | io_im_here |
|---|---|

Set window manager process state.

```
Entry Parameters:
     Register AL: 19
                  CL = im_here state
                       0 => manager not resident
                       1 => manager resident but not active
                       2 => manager resident and active
                       3 => leave state the same but
                            switch VC to top
                  DL = VC number to switch to top
                     = 0FFH means do not switch VC to top

 Returned Values:
              Window manager process state changed
              and/or specified VC is switched to
              the top and given the keyboard.
```

The process state can also be changed from a 1 to a 2 by the wake-up key, as described in the Concurrent CP/M User's Reference Guide for the IBM PC and PC XT. All other manager process state changes can be made only by using routine io_im_here.

| Function Code 20 | io_new_window |
|---|---|

Create a new window.

```
Entry Parameters:
     Register AL: 20
                  DL = virtual console number
                  CX = top left (row, column)
                  BX = bottom right (row, column)

 Returned Values:
              Window to specified VC is redrawn
              with new dimensions
```

Windows are specified by their inside corners. Rows are from 0 to 23, columns from 0 to 79. If the new size is the same as the the old size, then the physical image is updated from the virtual screen buffer. This provides the method of ensuring changes to the virtual screen buffer are correctly displayed on the physical console.

| Function Code 21 | io_cursor_view |
|---|---|
| Set the cursor tracking mode and the viewpoint. | |

```
Entry Parameters:
     Register AL: 21
               DL = virtual console number
               DH = cursor tracking mode
                    0 => fixed window (no tracking)
                    1 => track scrolling cursor
               CX = top left viewpoint corner
                    (row, column)

 Returned Values:
           Window's tracking mode and/or
           viewpoint are changed.
```

| Function Code 22 | io_wrap_column |
|---|---|
| Set virtual console wrap around column. | |

```
Entry Parameters:
     Register AL: 22
               DL = virtual console number
               CL = wrap column number

 Returned Values:
           Window's wrap column is changed
```

| Function Code 23 | io_full |
|---|---|
| Change window state. | |

```
Entry Parameters:
     Register AL: 23
               DL = virtual console number

 Returned Values:
           Window's state is changed from
           windowed to full or vice-versa.
```

| Function Code 24 | io_display |
|---|---|
| Specifies on which display the specified virtual console will be. | |
| Entry Parameters:<br>Register AL: 24<br>CL = 0 => monochrome<br>= 1 => color<br>DL = virtual console number<br><br>Returned Values:<br>None | |

## 1.4 Window Management Escape Sequences

The window management escape sequences provide a limited subset of window management functionality if you either cannot or do not want to link with assembly language routines that make back-door XIOS calls. These escape sequences all have the same basic format. In general, they are a list of bytes to be stored in AL, AH, BL, BH, CL, CH, DL, DH prior to making an XIOS call. The only XIOS calls that are supported via this escape sequence mechanism are listed in Table 1-4.

**Table 1-4. XIOS Calls for Escape Sequences**

| XIOS Call | Code | Function |
|---|---|---|
| io_switch | 7 | Switch screen to top but do not give it the keyboard. If screen is already on top, update the vc buffer with what is currently on the physical console. |
| io_statline | 8 | Display 80 character status line. |
| io_nstatline | 18 | New status line call with attributes. |
| io_im_here | 19 | Sets the window manager process state and changes which console is on top. |
| io_new_window | 20 | Sets a new console window. |
| io_cursor_view | 21 | Sets cursor track mode and viewpoint. |
| io_wrap_column | 22 | Sets the column for auto wrap-around. |

**Table 1-4. (continued)**

| XIOS Call | Code | Function |
|---|---|---|
| io_full | 23 | Toggles indicated VC from full to not full. |
| io_display | 24 | Sets which console the VC will be on. |

The escape sequences to access these calls use a parameter list of registers. There are always 10 bytes in the escape sequence. The last 8 bytes contain the desired values for the registers prior to an XIOS call. The exact escape sequence format is as follows:

ESC ! al ah bl bh cl ch dl dh

If you use these escape sequences in an application, you are giving up portability for the ability to use windows. Digital Research encourages OEMs to implement the same escape sequence conventions on their machines.

The program SAMPLE.C is included on the distribution diskette to demonstrate the technique for using Method 3 of console output.

End of Section 1

# Section 2
# Queue-driven Serial Communications

A Serial I/O device looks like a pair of queues to an application program. An application program can write to, or read from, a queue either conditionally or unconditionally. The details of synchronization and protocol are taken care of by the Input and Output RSP's, by the Interrupt Handler, and by the queue mechanism of Concurrent CP/M-86™ itself.

## 2.1 Theory of Operation

A character received by a Serial I/O Device generates an interrupt which activates the Interrupt Handler. The Interrupt Handler puts the character into the Circular Buffer, and then wakes up the Serial Input RSP. The Serial Input RSP reads one or more characters from the Circular Buffer, forms a message of the characters preceded by a count, and writes this message to the SerIn Queue. The received characters are now available to the application program.

When the application program wants to transmit characters, it can form a message of the same type as SerIn's (a string of characters preceded by a count), and write that message to the SerOut Queue. The Serial Output RSP reads this message and passes a pointer to the Interrupt Handler. The Interrupt Handler then transmits the message, one character at a time, and wakes up the Serial Output RSP when finished.

Each queue message is 17 bytes in length. The first byte, N, is a count indicating the number of valid data characters in the message. The next N bytes are the actual data in the order received or transmitted. This means that any message may contain from 1 to 16 bytes of actual data. The size of the input message is determined by the speed of the communications line and by system response time. Higher speeds of the received characters will produce longer messages on the average. The size of the output message is determined by the application program. Writing longer messages to the SerOut queue results in greater system efficiency, particularly at higher transmission speeds.

An Input/Output pair of RSP's, along with their associated queues, is automatically created during system initialization for each Serial Communiation Port installed. Serial Port 0 is defined as the card addressed at 03F8h, and is accessed through queues "SerIn0" and "SerOut0". Serial Port 1 is defined as the card addressed at 02F8h, and is accessed through queues "SerIn1" and "SerOut1". This numbering convention follows that of the Concurrent CP/M-86 CONFIG utility. If no Serial Communications Port is installed, then no Serial RSP or queue will be created. Figure 2-1 illustrates the system interface for queue-driven serial communication.

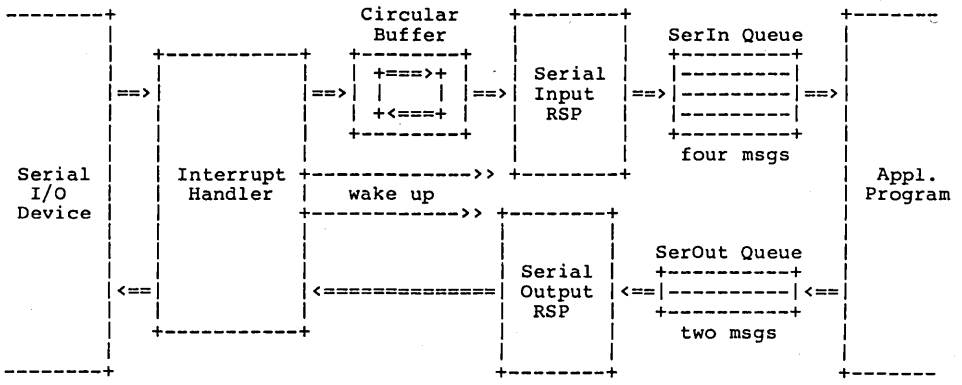


**Figure 2-1. Queue-driven Serial Communication Interface**

In addition to the SerIn and SerOut queues, there is another queue that is created for each port at system initialization. It is a mutual exclusion queue, named "MXSer0" for port 0, and "MXSer1" for port 1. These queues are designed to prevent access of more than one process to a single serial port. An application program reads its MX queue before accessing the Serial Data queues, and writes to its MX queue when it terminates.

There are limitations to the performance of Serial Communications under Concurrent CP/M-86. The primary limitation is encountered when using high speed communications while running other programs concurrently. There are times when the system "goes to sleep" for more than one high-baud-rate character time (Multi-sector disk access is an example). When this happens it is common to drop a received character or two, thus destroying the integrity of the incoming message. The other common problem is that the program receiving the data cannot consume it as fast as it is being sent. (Trying to send characters to the console at 9600 baud is an example of this.) Solutions to these problems must address the question of both high and low level communication protocol. Section 2.2 discusses communication protocol in detail.

The program TERM.A86 is included on the distribution diskette. This program demonstrates the use of queue-driven serial communication.

## 2.2 Protocol

The Interrupt Handler currently supports the following three types of handshaking protocol to synchronize communications between systems:

1. DTR/DSR is a hard-wired handshake using the Data Terminal Ready and Data Set Ready modem control lines.

2. RTS/CTS is a hard-wired handshake using the Request To Send and Clear To Send modem control lines.

3. XON/XOFF is a software handshake accomplished by sending an ASCII XON or XOFF character through the reverse channel.

A description of conditions and actions pertaining to all three protocols is presented below.

| | The Interrupt Handler, when receiving a character | | The Interrupt Handler, when transmitting a character | |
|---|---|---|---|---|
| Protocol | if buffer is almost full | if buffer is empty | will stop transmission | will resume transmission |
| DTR/DSR | Sets DTR = 0 | Sets DTR = 1 | if DSR = 0 | if DSR = 1 |
| RTS/CTS | Sets RTS = 0 | Sets RTS = 1 | if CTS = 0 | if CTS = 1 |
| XON/XOFF | Sends an XOFF | Sends an XON | if XOFF rec'd | if XON rec'd |

A given receive or transmit protocol can be set by an application program by writing a protocol message to the SerOut Queue. A protocol message is identified by a special code in the first byte of the queue message. The second byte of the message then sets the protocol. To set the receive protocol, the first byte of the message must be 0FEH. To set the transmit protocol, this byte must be 0FFH. In both cases, the second byte of the message contains a bit code for the protocol or protocols to be used. Table 2-1 describes the queue messages to set the various protocols.

**Table 2-1. Protocol and Queue Messages**

| Protocol | Queue Messages for Receive | for Transmit |
|---|---|---|
| <none> | 0FEH,00H | 0FFH,00H |
| DTR/DSR | 0FEH,01H | 0FFH,01H |
| RTS/CTS | 0FEH,02H | 0FFH,02H |
| DTR/DSR + RTS/CTS | 0FEH,03H | 0FFH,03H |
| XON/XOFF | 0FEH,04H | 0FFH,04H |
| DTR/DSR + XON/XOFF | 0FEH,05H | 0FFH,05H |
| RTS/CTS + XON/XOFF | 0FEH,06H | 0FFH,06H |
| DTR/DSR + RTS/CTS + XON/XOFF | 0FEH,07H | 0FFH,07H |

If the application program does not set the protocol, the default protocol for receive is <none> and the default protocol for transmit is DTR/DSR + XON/XOFF.

The program PROTOCOL.A86 is included on the distribution diskette. This program sets protocols for queue-driven serial communications.

## 2.3 Files and Sample Programs on the Distribution Diskette

Two window utilities, WINDOW and WMENU, are included on the distribution disk. The following files are required to generate the WINDOW and WMENU utilities (WINDOW.CMD and WMENU.CMD):

| | | |
|---|---|---|
| CAPARM.H | PDKEEP.A86 | WINDOW.C |
| FIELDS.H | WMCA.A86 | WMCOLOR.C |
| PORTAB.H | WWCALL.A.86 | WMENU.C |
| | | WMEXEC.C |
| | | WMFTD.C |
| | | WMUTIL.C |
| | | WMWINDOW.C |
| | | WMWRITE.C |

A submit file, WINDOW.SUB, is included to generate WMENU.CMD and WINDOW.CMD from the preceding files.

The sample program, SAMPLE.C, uses the .H files listed above along with WMCA.A86 and WWCALL.A86.

The sample program TERM.A86 on your distribution diskette demonstrates the use of queue-driven serial communication. The sample program PROTOCOL.A86 demonstrates protocols for queue-driven serial communication.

The Digital Research C™ compiler is required to compile the programs with filetype C. ASM-86™ is needed to assemble the programs with the A86 filetype. GENCMD can be used to generate the CMD files.

End of Section 2

# NOTES

# NOTES

# Reader Comment Card

We welcome your comments and suggestions. They help us provide you with better product documentation.

Date _____

1. What sections of this manual are especially helpful?

2. What suggestions do you have for improving this manual? What information is missing or incomplete? Where are examples needed?

3. Did you find errors in this manual? (Specify section and page number.)

Concurrent CP/M™ with Windows Operating System
Technical Notes for the IBM PC & PC XT
First Edition: February 1984
4007-1010-001

From: _____

_____

_____

_____

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS / PERMIT NO. 182 / PACIFIC GROVE, CA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL RESEARCH®

**Attn: Publications Production**

P.O. BOX 579

PACIFIC GROVE, CA 93950-9987