x3270-script Manual Page

Contents

Name

Scripting Facilities for x3270, c3270 and s3270

Synopsis

x3270 -script [*x3270-options*]
x3270 -socket [*x3270-options*]
x3270 -scriptport ***port*** [*x3270-options*]
c3270 -socket [*c3270-options*]
c3270 -scriptport *port* [*s3270-options*]
s3270 [*s3270-options*]
Script (*command* [*,arg...*])

Description

The **x3270** scripting facilities allow the interactive 3270 emulators **x3270** and **c3270** to be operated under the control of another program, and forms the basis for the script-only emulator **s3270**.

There are four basic scripting methods. The first is the **peer script** facility, invoked by the **x3270 -script** switch, and the default mode for **s3270**. This runs the emulator as a child of another process. Typically this would be a script using *expect*(1), *perl*(1), or the co-process facility of the Korn Shell *ksh*(1). In this mode, the emulator process looks for commands on its standard input, and places the responses on standard output.

The second method is the **child script** facility, invoked by the emulator's **Script** action. This runs a script as a child process of the emulator. The child has access to pipes connected to the emulator; the emulator looks for commands on one pipe, and places the responses on the other. The file descriptor of the pipe for commands to the emulator is passed in the environment variable X3270INPUT (e.g., the text string "7" if the file descriptor is 7); the file descriptor of the pipe for responses from the emulator is passed in the environment variable X3270OUTPUT.

The third method uses a TCP socket. The **-scrpiptport** command-line option causes the emulator to bind a socket to the specified port (on the IPv4 loopback address, 127.0.0.1). The emulator accepts TCP connections on that port. Multiple commands and responses can be sent over each connection.

The fourth method uses a Unix-domain socket. The **-socket** command-line option causes the emulator to create a Unix-domain stream socket named **/tmp/x3sck.***pid*. The emulator accepts connections to that socket. Multiple commands and responses can be sent over each connection.

It is possible to nest the methods. For example, a peer or TCP socket script can invoke the **Script** action. The calling script will be resumed when the nested script completes.

Commands are emulator *actions*; the syntax is the same as for the right-hand side of an **x3270** or **c3270** keymap. Unlike translation tables, action names are case-insensitive, can be uniquely abbreviated, and the parentheses may be omitted if there are no parameters. Any input line that begins with **#** or **!** is treaded as a comment and will be ignored.

Any emulator action may be specified. Several specific actions have been defined for use by scripts, and the behavior of certain other actions (and of the emulators in general) is different when an action is initiated by a script.

Some actions generate output; some may delay completion until the certain external events occur, such as the host unlocking the keyboard. The completion of every command is marked by a two-line message. The first line is the current status of the emulator, documented below. If the command is successful, the second line is the string "ok"; otherwise it is the string "error".

Status Format

The status message consists of 12 blank-separated fields:

1 Keyboard State

If the keyboard is unlocked, the letter **U**. If the keyboard is locked waiting for a response from the host, or if not connected to a host, the letter **L**. If the keyboard is locked because of an operator error (field overflow, protected field, etc.), the letter **E**.

2 Screen Formatting

If the screen is formatted, the letter **F**. If unformatted or in NVT mode, the letter **U**.

3 Field Protection
: If the field containing the cursor is protected, the letter **P**. If unprotected or unformatted, the letter **U**.

4 Connection State
: If connected to a host, the string **C(***hostname***)**. Otherwise, the letter **N**.

5 Emulator Mode
: If connected in 3270 mode, the letter **I**. If connected in NVT line mode, the letter **L**. If connected in NVT character mode, the letter **C**. If connected in unnegotiated mode (no BIND active from the host), the letter **P**. If not connected, the letter **N**.

6 Model Number (2-5)

7 Number of Rows
: The current number of rows defined on the screen. The host can request that the emulator use a 24x80 screen, so this number may be smaller than the maximum number of rows possible with the current model.

8 Number of Columns
: The current number of columns defined on the screen, subject to the same difference for rows, above.

9 Cursor Row
: The current cursor row (zero-origin).

10 Cursor Column
: The current cursor column (zero-origin).

11 Window ID
: The X window identifier for the main **x3270** window, in hexadecimal preceded by **0x**. For **s3270** and **c3270**, this is zero.

12 Command Execution Time
: The time that it took for the host to respond to the previous commnd, in seconds with milliseconds after the decimal. If the previous command did not require a host response, this is a dash.

Differences

When an action is initiated by a script, the emulators behave in several different ways:

If an error occurs in processing an action, the usual pop-up window does not appear. Instead, the text is written to standard output.

If end-of-file is detected on standard input, the emulator exits. (A script can exit without killing the emulator by using the **CloseScript** action, below.) Note that this applies to peer scripts only; end-of-file on the pipe connected to a child script simply causes the pipes to be closed and the **Script** action to complete.

The **Quit** action always causes the emulator to exit. (When called from the keyboard, it will exit only if not connected to a host.)

Normally, the AID actions (**Clear**, **Enter**, **PF**, and **PA**) will not complete until the host unlocks the keyboard. If the parameter to a **String** action includes a code for one these actions, it will also wait for the keyboard to unlock before proceeding.

The **AidWait** toggle controls with behavior. When this toggle is set (the default), actions block as described above. When the toggle is clear, AID actions complete immediately. The **Wait(Output)** action can then be used to delay a script until the host changes something on the screen, and the **Wait(Unlock)** action can be used to delay a script until the host unlocks the keyboard, regardless of the state of the **AidWait** toggle.

Note that the **Script** action does not complete until end-of-file is detected on the pipe or the **CloseScript** action is called by the child process. This behavior is not affected by the state of the **AidWait** toggle.

Basic Programming Strategies

3270 session scripting can be more difficult than other kinds of scripting, because it can be hard to tell when the host is finished processing a command. There is a well-defined 3270 Data Stream facility for doing this: The emulator locks the keyboard when it sends the host an AID, and the later host unlocks the keyboard. The emulator supports this facility directly by not allowing an AID action to complete until the keyboard is unlocked. Unfortunately, some hosts and some host applications unlock the keyboard as soon as they begin processing the command, instead of after it is finished. A human operator can see on the screen when the command is finished (e.g., when a READY prompt is displayed), but it can be difficult for a script to do this. For such early-unlock hosts, the only option in a script is to poll the screen until it can determine that the command is complete.

Another complication is that host I/O and script operation are asynchronous. That is, the host can update the screen at any time, even between actions that are reading the screen contents, so a script can get inconsistent results. Assistance for this problem is provided by the **Snap** action. The **Snap(Save)** action saves a snapshot of the screen in a special buffer. Then the script can use **Snap** variants of the **Ascii** and **Ebcdic** actions (**Snap(Ascii)** and **Snap(Ebcdic)**) to query the saved buffer -- which the host cannot modify -- to get the data it wants. Finally, **Snap(Wait Output)** blocks the script until the host modifies the screen, specifically since the last call to **Snap(Save)**. Thus a script can poll the screen efficiently by writing a loop that begins with **Snap(Save)** and ends with **Snap(Wait Output)**.

Script-Specific Actions

The following actions have been defined or modified for use with scripts. (Note that unlike the display on the status line, *row* and *col* coordinates used in these actions use [0,0] as their origin at the upper left, not [1,1]).

AnsiText

Outputs whatever data that has been output by the host in NVT mode since the last time that **AnsiText** was called. The data is preceded by the string "data: ", and has had all control characters expanded into C backslash sequences.

This is a convenient way to capture NVT mode output in a synchronous manner without trying to decode the screen contents.

Ascii(*row*,*col*,*rows*,*cols*)
Ascii(*row*,*col*,*length*)
Ascii(*length*)
Ascii

Outputs an ASCII text representation of the screen contents. Each line is preceded by the string "data: ", and there are no control characters.

If four parameters are given, a rectangular region of the screen is output. (Note that the row and column are zero-origin.)

If three parameters are given, *length* characters are output, starting at the specified zero-origin row and column.

If only the *length* parameter is given, that many characters are output, starting at the cursor position.

If no parameters are given, the entire screen is output.

The EBCDIC-to-ASCII translation and output character set depend on the both the emulator character set (the **-charset** option) and the locale. UTF-8 and certain DBCS locales may result in multi-byte expansions of EBCDIC characters that translate to ASCII codes greater than 0x7f.

AsciiField

Outputs an ASCII text representation of the field containing the cursor. The text is preceded by the string "data: ".

Connect(*hostname*)

Connects to a host. The command does not return until the emulator is successfully connected in the proper mode, or the connection fails.

CloseScript(*status*)

Causes the emulator to stop reading commands from the script. This is useful to allow a peer script to exit, with the emulator proceeding interactively. (Without this command, the emulator would exit when it detected end-of-file on standard input.) If the script was invoked by the **Script** action, the optional *status* is used as the return status of **Script**; if nonzero, **Script** will complete with an error, and if this script was invoked as part of login through the **ibm_hosts** file, the connection will be broken.

ContinueScript([*param*])

Allows a script that is waiting in a **PauseScript** action, below, to continue. The optional *param* string is output by the **PauseScript** action.

Disconnect
: Disconnects from the host.

Ebcdic(*row*,*col*,*rows*,*cols*)
Ebcdic(*row*,*col*,*length*)
Ebcdic(*length*)
Ebcdic
: The same function as **Ascii** above, except that rather than generating ASCII text, each character is output as a 2-digit or 4-digit hexadecimal EBCDIC code.

EbcdicField
: The same function as **AsciiField** above, except that it generates hexadecimal EBCDIC codes.

Info(*message*)
: In x3270, pops up an informational message. In c3270 and wc3270, writes an informational message to the OIA (the line below the display). Not defined for s3270 or tcl3270.

Expect(*text*[,*timeout*])
: Pauses the script until the specified *text* appears in the data stream from the host, or the specified *timeout* (in seconds) expires. If no *timeout* is specified, the default is 30 seconds. *Text* can contain standard C-language escape (backslash) sequences. No wild-card characters or pattern anchor characters are understood. **Expect** is valid only in NVT mode.

MoveCursor(*row*,*col*)
: Moves the cursor to the specified zero-origin coordinates.

PauseScript
: Stops a script until the **ContinueScript** action, above, is executed. This allows a script to wait for user input and continue. Outputs the single parameter to **ContinueScript**, if one is given.

PrintText([**command**,]*filter*)
: Pipes an ASCII representation of the current screen image through the named *filter*, e.g., **lpr**.

PrintText([**html**,][**append**,][**replace**,]**file**,*filename*)
: Saves the current screen contents in a file. With the **html** option, saves it as HTML, otherwise saves it as plain ASCII. The **append** option (the default) causes the data to be appended to the file if it already exists. The **replace** option causes the file to be overwritten instead.

PrintText(**html,string**)
: Returns the current screen contents as HTML.

Query(*keyword*)
: Returns state information. Keywords are:

Keyword	Output
BindPluName	BIND PLU returned by the host
ConnectionState	TN3270/TN3270E mode and submode
CodePage	Host code page

Cursor	Cursor position (row col)
Formatted	3270 format state (formatted or unformatted)
Host	Host name and port
LocalEncoding	Local character encoding
LuName	Host name LU name
Model	3270 model name (IBM-327x-n)
ScreenCurSize	Current screen size (rows cols)
ScreenMaxSize	Maximum screen size (rows cols)
Ssl	SSL state (secure or not-secure) and host validation state (host-verified or host-unverified)

Without a *keyword*, **Query** returns each of the defined attributes, one per line, labeled by its name.

ReadBuffer(Ascii)

Dumps the contents of the screen buffer, one line at a time. Positions inside data fields are generally output as 2-digit hexadecimal codes in the current display character set. If the current locale specifies UTF-8 (or certain DBCS character sets), some positions may be output as multi-byte strings (4-, 6- or 8-digit codes). DBCS characters take two positions in the screen buffer; the first location is output as a multi-byte string in the current locale codeset, and the second location is output as a dash. Start-of-field characters (each of which takes up a display position) are output as **SF(aa=nn[,...])**, where *aa* is a field attribute type and *nn* is its value.

Attribute	Values
c0 basic 3270	20 protected
	10 numeric
	04 detectable
	08 intensified
	0c non-display
	01 modified
41 highlighting	f1 blink
	f2 reverse
	f4 underscore
	f8 intensify
42 foreground	f0 neutral black
	f1 blue
	f2 red
	f3 pink
	f4 green
	f5 turquoise

	f6 yellow
	f7 neutral white
	f8 black
	f9 deep blue
	fa orange
	fb purple
	fc pale green
	fd pale turquoise
	fe grey
	ff white
43 character set	f0 default
	f1 APL
	f8 DBCS

Extended attributes (which do not take up display positions) are output as **SA(aa=nn)**, with *aa* and *nn* having the same definitions as above (though the basic 3270 attribute will never appear as an extended attribute).

In addition, NULL characters in the screen buffer are reported as ASCII character 00 instead of 20, even though they should be displayed as blanks.

ReadBuffer(**Ebcdic**)
Equivalent to **ReadBuffer**(**Ascii**), but with the data fields output as hexadecimal EBCDIC codes instead. Additionally, if a buffer position has the Graphic Escape attribute, it is displayed as **GE(*xx*)**.

Script(*path*[,arg...])
Runs a child script, passing it optional command-line arguments. *path* must specify an executable (binary) program: the emulator will create a new process and execute it. If you simply want the emulator to read commands from a file, use the **Source** action.

Snap
Equivalent to **Snap**(**Save**) (see **below**).

Snap(**Ascii**,...)
Performs the **Ascii** action on the saved screen image.

Snap(**Cols**)
Returns the number of columns in the saved screen image.

Snap(**Ebcdic**,...)
Performs the **Ebcdic** action on the saved screen image.

Snap(**ReadBuffer**)
Performs the **ReadBuffer** action on the saved screen image.

Snap(Rows)
Returns the number of rows in the saved screen image.

Snap(**Save**)
Saves a copy of the screen image and status in a temporary buffer. This copy can be queried with other **Snap** actions to allow a script to examine a consistent screen image, even when the host may be

changing the image (or even the screen dimensions) dynamically.

Snap(**Status**)
Returns the status line from when the screen was last saved.

Snap(**Wait**[,*timeout*],**Output**)
Pauses the script until the host sends further output, then updates the snap buffer with the new screen contents. Used when the host unlocks the keyboard (allowing the script to proceed after an **Enter**, **PF** or **PA** action), but has not finished updating the screen. This action is usually invoked in a loop that uses the **Snap**(**Ascii**) or **Snap**(**Ebcdic**) action to scan the screen for some pattern that indicates that the host has fully processed the last command.

The optional *timeout* parameter specifies a number of seconds to wait before failing the **Snap** action. The default is to wait indefinitely.

Source(*file*)
Read and execute commands from *file*. Any output from those commands will become the output from **Source**. If any of the commands fails, the **Source** command will *not* abort; it will continue reading commands until EOF.

Title(*text*)
Changes the x3270 window title to *text*.

Transfer(*keyword=value*,...)
Invokes IND$FILE file transfer. See **FILE TRANSFER** below.

Wait([*timeout*,] **3270Mode**)
Used when communicating with a host that switches between NVT mode and 3270 mode. Pauses the script or macro until the host negotiates 3270 mode, then waits for a formatted screen as above.

The optional *timeout* parameter specifies a number of seconds to wait before failing the **Wait** action. The default is to wait indefinitely.

For backwards compatibility, **Wait(3270)** is equivalent to **Wait**(**3270Mode**)

Wait([*timeout*,] **Disconnect**)
Pauses the script until the host disconnects. Often used to after sending a *logoff* command to a VM/CMS host, to ensure that the session is not unintentionally set to **disconnected** state.

The optional *timeout* parameter specifies a number of seconds to wait before failing the **Wait** action. The default is to wait indefinitely.

Wait([*timeout*,] **InputField**)
A useful utility for use at the beginning of scripts and after the **Connect** action. In 3270 mode, waits until the screen is formatted, and the host has positioned the cursor on a modifiable field. In NVT mode, waits until the host sends at least one byte of data.

The optional *timeout* parameter specifies a number of seconds to wait before failing the **Wait** action. The default is to wait indefinitely.

For backwards compatibility, **Wait** is equivalent to **Wait**(**InputField**).

Wait([*timeout*,] **NVTMode**)
: Used when communicating with a host that switches between 3270 mode and NVT mode. Pauses the script or macro until the host negotiates NVT mode, then waits for a byte from the host as above.

  The optional *timeout* parameter specifies a number of seconds to wait before failing the **Wait** action. The default is to wait indefinitely.

  For backwards compatibility, **Wait**(**ansi**) is equivalent to **Wait**(**NVTMode**).

Wait([*timeout*,] **Output**)
: Pauses the script until the host sends further output. Often needed when the host unlocks the keyboard (allowing the script to proceed after a **Clear**, **Enter**, **PF** or **PA** action), but has not finished updating the screen. Also used in non-blocking AID mode (see **DIFFERENCES** for details). This action is usually invoked in a loop that uses the **Ascii** or **Ebcdic** action to scan the screen for some pattern that indicates that the host has fully processed the last command.

  The optional *timeout* parameter specifies a number of seconds to wait before failing the **Wait** action. The default is to wait indefinitely.

Wait([*timeout*,] **Unlock**)
: Pauses the script until the host unlocks the keyboard. This is useful when operating in non-blocking AID mode (**toggle AidWait clear**), to wait for a host command to complete. See **DIFFERENCES** for details).

  The optional *timeout* parameter specifies a number of seconds to wait before failing the **Wait** action. The default is to wait indefinitely.

Wait(*timeout*, **Seconds**)
: Delays the script *timeout* seconds. Unlike the other forms of **Wait**, the timeout is not optional.

WindowState(*mode*)
: If *mode* is **Iconic**, changes the x3270 window into an icon. If *mode* is **Normal**, changes the x3270 window from an icon to a normal window.

File Transfer

The **Transfer** action implements **IND$FILE** file transfer. This action requires that the **IND$FILE** program be installed on the IBM host, and that the 3270 cursor be located in a field that will accept a TSO or VM/CMS command.

Because of the complexity and number of options for file transfer, the parameters to the **Transfer** action take the unique form of *option*=*value*, and can appear in any order. Note that if the *value* contains spaces (such as a VM/CMS file name), then the entire parameter must be quoted, e.g., "HostFile=xxx foo a". The options are:

Option	Required?	Default	Other Values
Direction	No	receive	send
HostFile	Yes		
LocalFile	Yes		
Host	No	tso	vm, cics
Mode	No	ascii	binary
Cr	No	remove	add, keep
Remap	No	yes	no
Exist	No	keep	replace, append
Recfm	No		fixed, variable, undefined
Lrecl	No		
Blksize	No		
Allocation	No		tracks, cylinders, avblock
PrimarySpace	Sometimes		
SecondarySpace	No		
Avblock	Sometimes		
BufferSize	No	4096	

The option details are as follows.

Direction
: **send** to send a file to the host, **receive** to receive a file from the host.

HostFile
: The name of the file on the host.

LocalFile
: The name of the file on the local workstation.

Host
: The type of host (which dictates the form of the **IND$FILE** command): **tso** (the default), **vm** or **cics**.

Mode
: Use **ascii** (the default) for a text file, which will be translated between EBCDIC and ASCII as necessary. Use **binary** for non-text files.

Cr
: Controls how **Newline** characters are handled when transferring **Mode=ascii** files. **remove** (the default) strips **Newline** characters in local files before transferring them to the host. **add** adds **Newline** characters to each host file record before transferring it to the local workstation. **keep** preserves **Newline** characters when transferring a local file to the host.

Remap
: Controls text translation for **Mode=ascii** files. The value **yes** (the default) causes x3270 to remap the text to ensure maximum compatibility between the workstation's character set and encoding

and the host's EBCDIC code page. The value **no** causes x3270 to pass the text to or from the host as-is, leaving all translation to the **IND$FILE** program on the host.

Exist
: Controls what happens when the destination file already exists. **keep** (the default) preserves the file, causing the **Transfer** action to fail. **replace** overwrites the destination file with the source file. **append** appends the source file to the destination file.

Recfm
: Controls the record format of files created on the host. (TSO and VM hosts only.) **fixed** creates a file with fixed-length records. **variable** creates a file with variable-length records. **undefined** creates a file with undefined-length records (TSO hosts only). The **Lrecl** option controls the record length or maximum record length for **Recfm=fixed** and **Recfm=variable** files, respectively.

Lrecl
: Specifies the record length (or maximum record length) for files created on the host. (TSO and VM hosts only.)

Blksize
: Specifies the block size for files created on the host. (TSO and VM hosts only.)

Allocation
: Specifies the units for the **PrimarySpace** and **SecondarySpace** options: **tracks**, **cylinders** or **avblock**. (TSO hosts only.)

PrimarySpace
: Primary allocation for a file. The units are given by the **Allocation** option. Required when the **Allocation** is specified as something other than **default**. (TSO hosts only.)

SecondarySpace
: Secondary allocation for a file. The units are given by the **Allocation** option. (TSO hosts only.)

Avblock
: Average block size, required when **Allocation** specifies **avblock**. (TSO hosts only.)

BufferSize
: Buffer size for DFT-mode transfers. Can range from 256 to 32768. Larger values give better performance, but some hosts may not be able to support them.

There are also resources that control the default values for each of the file transfer parameters. These resources have the same names as the **Transfer** keywords, but with **ft** prepended. E.g., the default for the **Mode** keyword is the **x3270.ftMode** resource.

See Also

expect(1)
perl(1)
ksh(1)
x3270(1)

x3270if(1)
c3270(1)
s3270(1)

Version

Version 3.5ga8

*This HTML document and the accompanying troff document were generated with a set of write-only **m4** macros and the powerful **vi** editor.*
Last modified 09 May 2016.