

Appendix: Writing external DAVEX commands

DAL Systems 20-Feb-90 Version 1.25

This appendix is for assembly-language programmers. It explains the format of Davex external commands and the resources available to them.

Three files are provided that should be "put" (included) by external commands; these files are in Merlin format and will have to be modified some if you are using EDASM or some other assembler. (1) "globals.s" defines entry points and locations provided by Davex, (2) "apple.globals.s" defines entry points in the Apple ROM and some locations on zero page and page 3, and (3) "mli.globals.s" defines ProDOS 8 command numbers, error codes, and global-page locations.

"xc.s" is the source code for an empty external command; the best way to start a new command is by making a copy of "xc.s". Source code for the "du" external command is also provided as an example.

An external Davex command is assembled to run below \$B000. External commands have a fixed (roughly) ending address to allow Davex to grow without overlapping previously-assembled external commands. Command files have filetype \$2E, auxiliary type \$8001. (For compatibility with old versions of Davex, commands may also have filetype BIN.)

External commands can have all the same kinds of parameters built-in commands can have.

Here is the format for an external command:

```
$60      (RTS)
$EE
$EE
$xx      version # of command ($34 = Version 3.4) (1 byte)
          (recommended: use versions less than 1.0 for
incomplete versions)
$xx      minimum Davex version required (1 byte)
          (use the version number for the Davex you are
working with, unless you're sure your command works with
earlier versions) [see auxiliary version nibble below]
$xx      command characteristics (1 byte)
          7: requires 40-col screen
          6: requires 80-col screen
          5: requires //e or IIgs
          4: requires //c
          3: requires IIgs
          2..0: reserved; use 0
```

2-byte pointer to ASCII description text (or 0)

Description must be in first 512 bytes of the object file and must be preceded by a length byte. The author's name or other identification should appear in

this description for commands not distributed by DAL Systems. (The 'what' command displays these descriptions--be sure to test it on your commands.)

load address (2 bytes)

The address at which this file must be loaded-- typically an exact page boundary that makes the code end shortly before \$B000.

execution address (2 bytes)

Davex will JSR to the address stored here once the command is loaded at its load address.

auxiliary minimum Davex version required [v1.22+]

The low nibble of this byte is an extension of the "minimum Davex version" byte above. For example, a command that requires Davex version 1.82 or greater would have \$02 in this byte. (The high nibble of this byte is reserved and should be 0 for now.)

3 bytes reserved for future extensions of the external command format (FILL WITH 0)

parameter table (see below) (2p+2 bytes; p=number of parameters)

2 bytes for each parameter; then 2 bytes of \$00

Davex loads the external command at its load address (after checking that it would not overlap memory used by Davex). Davex then evaluates any parameters and calls the command's execution address AT LEAST once. If the command has any 'wildpath' parameters, Davex expands the wildcards and calls the command once for each file that matches the wildcard, without reloading the external command. (Even if your command does not take wildcards, do not assume that it will be reloaded whenever it is used: the 'rep' command may execute your command repeatedly without reloading it.)

When the external command finishes its work, it will normally RTS back to Davex. (If an error occurs, it might JMP to xerr or to xProDOS_err instead--see below.)

The parameter table is a list of two-byte entries. A double-zero entry marks the end of the table. The first byte of each pair is the option character for one parameter (use 0 for parameters without any associated "-" character). The second byte is the type.

If one wildcard pathname is allowed, it must be the first parameter in the table. If two wildcard pathnames are allowed, they must be the first two parameters in the table-- wildcard matching will take place on the first name, and the text matching the wildcard will be substituted into the second parameter whenever a wildcard character appears. All of the wildcard processing is invisible to external commands.

The option characters in the parameter table must be in lower case (if they are letters) and must have their high bits on.

Note that all "required" parameters (parameters that are not associated with a dash and a character) must come before all other parameters in the table, even though the user no longer needs to type them in that order.

Also, a required pathname or string parameter may contain zero characters. From the user's point of view, the parameter is optional--but it is treated internally as a string with no characters. The difference is important when your command decides how to act on the parameter's value.

If you do not want Davex to display each wildcard match before calling your command, use a NOP as the first executable byte of your code. Commands that always incorporate the pathname(s) passed to them into their output may want to do this. (The 'what' command is a good example.)

Note: When Davex prints a wildcard match, it prints it to the SCREEN, never to a file or to the printer. If your command would be useful for printing "tables" of information (something like 'size' and 'what' do), you will probably want to use the NOP option and have your command print the filenames or pathnames itself.

Parameter types (defined in "globals"):

t_nil	no value associated with option character
t_int1	1-byte integer (Y)
t_int2	2-byte integer (XY) (X=high, Y=low)
t_int3	3-byte integer (AXY) (A=highest, Y=lowest)
t_path	ProDOS pathname (AY; X=file type given after name)
t_wildpath	ProDOS pathname allowing wildcards (AY)
t_string	string value (AY)
t_yesno	y/n (A: 0=no, \$80=yes)
t_ftype	file type (A)
t_devnum	device number (A) (example: ".62" = slot 6, drive 2):
	A=\$E0 (\$80+\$60)

For string and pathname values, a pointer is passed in A and Y. The data pointed to is a length-prefixed string, suitable for use in a ProDOS parameter block.

--Resources available to external commands--

External commands may use 'filebuff', 'filebuff2', and 'filebuff3', defined in GLOBALS; each one is \$400 bytes long.

32 bytes of zero page are reserved for XCs at 'xczpage'.

The high bit of 'xspeech' is on when a speech synthesizer is being used.

XCs may use standard Monitor ROM routines for output (but not for input). The following entry points into Davex are defined in the Globals file. ASSUME ALL REGISTERED ARE SCRAMBLED, EXCEPT AS DOCUMENTED BELOW.

`xgetparm_n` -- get value of parameter number A

Use this subroutine to get the values of required parameters (ones with a 0 for the first byte of their pair in the parameter table). Before calling this routine, load the A register with a parameter number (the first parameter is 0).

For example, a "rename" command would LDA #0, JSR `xgetparm_n` to get the value of the first parameter. Then it would LDA #1, JSR `xgetparm_n` to get the value of the second parameter.

NOTE: When an external command gets control, "LDA #0; JSR `xgetparm_n`" has just been done.

`xgetparm_ch` -- get value of parameter for option char in A

Use this routine to get the value of an optional parameter (one that has a character in the first byte of its pair in the parameter table). Load the A register with the character before calling this routine. The character should be lowercase (if it's a letter) and have its high bit ON.

If the parameter in question was not given on the command line, this routine will return with the carry flag set (SEC). Otherwise the carry will be clear and the value of the parameter will be in the appropriate registers (A,X,Y--see table of parameter types above).

`xmess`

Prints an inline message--ASCII text, followed by a \$00, follows the JSR to this subroutine.

`xprint_ftype`

Takes filetype code in A and prints a three-character filetype name, or \$xx if the filetype is not known to Davex (the lists of known filetypes are stored in the %config file and in Davex itself, and the "ftype" command can be used to view and edit the user's list).

`xprint_access`

Takes a ProDOS access byte and prints:

`rwndIB`

Only the letters corresponding to bits set in the access byte are printed; blanks are printed for the others (r=read, w=write, n=rename, d=delete, I=invisible, B=needs backup).

`xprdec_2`

Prints a 2-byte value in decimal. AY contains the value (A=high byte). No characters are printed before or after the number.

xprdec_3

Prints a 3-byte value in decimal. The number must be stored in xnum (lowest byte), xnum+1 (middle byte), and xnum+2 (highest byte). No characters are printed before or after the number.

xprdec_pady

Prints a decimal number from NUM (3 bytes) right-justified in a field of Y+1 characters

xprdec_pad

Same as xprdec_3, except the number is right-justified in a field of 7 characters.

xprint_path

Prints (in lowercase) a length-prefixed string pointed to by AY.

xbuild_local

AY must point to a partial pathname; builds a complete pathname by appending to the "%" directory name. This will locate the %config file, for example. Returns AY pointing to the complete path. (If you call this routine more than once, note that the same memory will be used to store the pathname, so the previous name will be erased.)

xprint_sd

Entry: A=device number; prints: .sd, where s and d are the slot and drive of the given device number.

xprint_drvr [input functions in Davex 1.25+]

Provides calls to open, close, write, and poll character devices. Entry: X = function. Other parameters depend on the function. For convenience, ProDOS call numbers are used (as defined in MLI.GLOBALS.S).

Slots are opened -independently- for input and output. Input is supported for Pascal devices, but not for parallel cards.

X = mli_open: open a slot for output
Input: A=slot number (0 for default)
Output: CLC, A=reference number
SEC, A=error code (for xProDOS_err)

X = mli_open-\$80: open a slot for input
Input: A=slot number
Output: CLC, A=reference number
SEC, A=error code (for xProDOS_err)

X = mli_close: close a slot for output

Input: Y=reference number
Output: CLC = successful
SEC, A=error code (for xProDOS_err)

X = mli_close-\$80: close a slot for input
Input: Y=reference number
Output: CLC = successful
SEC, A=error code (for xProDOS_err)

X = mli_write: send a character (7 bits significant)
Input: A=character to be written
Y=reference number from the open
Output: CLC = successful
SEC, A=error code (for xProDOS_err)

X = mli_write-\$80: send a character (all 8 bits significant)
inputs and outputs as for mli_write, above

X = mli_read: see if device is ready to accept output
Input: Y=reference number
Output: CLC = successful
Bit 0 of A is 1 if device is ready
to receive
output; other bits are undefined
SEC, A=error code (for xProDOS_err)

X = mli_read-\$80
Input: Y=reference number
Output: CLC, A=character successfully read
SEC, A=0 if no character ready
SEC, A>0 if error (for xProDOS_err)

xredirect

Controls suspension of I/O redirection (multiple levels of suspension are allowed; one "restore" is required for each "suspend").

Input in A:
0: determine current suspension level
1: suspend I/O redirection
-1: (\$FF): restore I/O redirection

Output in A:
N flag = bit 7 = 1 if output is being redirected
V flag = bit 6 = 1 if input is being redirected

xpercent

Takes two 3-byte values: one in AXY and one in xnum.
Returns (in A) the percentage that AXY is of xnum (3 bytes).

xyesno

Prints '? (y/n)' and waits for a Y or N to be typed.
Returns:
No: A=\$00, Z flag=1 (BEQ will be taken)
Yes: A=\$80, Z flag=0 (BNE will be taken)

See xredirect notes in xyesno2 description.

xyesno2 [Davex v1.2+]

Just like xyesno, except that the SPACE and RETURN keys are also accepted. Before calling this routine, load the A register with a 'y' or an 'n'. If the user types a SPACE or RETURN, it is translated into the character you passed.

xyesno2 should be used when there is a clear and safe default choice at a yes/no question. The default should never be destructive! If there is no clear default choice, don't try to outguess the user; just use xyesno.

To asking a yes/no question, you should call xredirect with A=1 to suspend any active I/O redirection, print the prompt, call xyesno or xyesno2, call xredirect with A=-1 (\$FF), and then act on the answer to the question. If you don't call xredirect, the question may get printed or sent to a disk file, and the answer to the question may come from an exec file!

```
        lda #1
        jsr xredirect
        jsr xmess
        asc "Okay to detonate mouse"
        dfb 0

        lda #"n"
        jsr xyesno2

        php                ;save Z flag for BEQ/BNE
        lda #-1
        jsr xredirect
        plp

        beq TheySaidNo
        ...
```

xgetln

get an input line and place it in string, [NOT string2; this was documented wrong before] zero-terminated (there is also a length byte at string-1). Returns SEC if input was cancelled by Ctrl-X. This is the same string that Davex uses for the command line, so a command that uses this call must preserve and restore the contents of this buffer [256 bytes starting at string-1 (this is actually overkill by a few bytes)]. Note that the up and down arrows *will* allow the user to scroll through the command history if this call is used. A BETTER GETLN ROUTINE WILL BE AVAILABLE IN DAVEX 1.3:

xgetln2 [PLANNED FOR DAVEX 1.3]

Input: AY = address of input buffer (provided by your

command)

X = length of the buffer

Output: length-prefixed, zero-terminated string in buffer

xgetln2 reads lines of text from an input device. Input will often come from the keyboard, but it can come from an exec file or a peripheral device. If you want to force input to come from the keyboard, use xredirect to suspend I/O redirection before calling xgetln2 and to restore it afterwards. (If you print a prompt before getting the input, be sure to print the prompt AFTER suspending I/O redirection.)

Unlike xgetln, the up and down arrows (for history scrolling) are disabled during xgetln2.

Note that the maximum number of characters in the input string is two less than the size of the buffer, since one byte is used for the length of the string and one \$00 byte marks the end of the string. xgetln2's behavior is undefined if it is called with X less than 2.

xbell

Sound a warning bell (a ProDOS-style "blat" or a system "beep", depending on "config -b").

xdowncase

If character in A is a capital letter, changes it into a lowercase letter (always sets bit 7). X and Y are preserved; A is preserved or capitalized. This routine always turns on the high bit of the character in A.

xplural

Takes two-byte value in AY and prints 's' if the value is not equal to 1. Let's not have any more "1 files found" messages!

xcheck_wait

Returns with SEC if the user has pressed ESC. This is a SOFT ABORT if your command supports it; wildcard expansion and further command-line processing, if any, will continue. If the user hits Ctrl-C or Apple-period, this routine will print "*** aborted" and will clean up and return to the command prompt.

The user can also PAUSE and single-step the screen by hitting SPACE. This routine will call poll_io (for print spooling) while the screen is frozen. Also, Apple-H will do a screen dump (except on II+).

NOTE: If an external command calls xcheck_wait, it should do it exactly once per line printed. Test your command for reasonable behavior by single-stepping the output with the space bar.

xpr_date_ay

Takes standard ProDOS date word in AY and prints date in the form dd-mmm-yy. If AY=0, prints "<no date>" instead. Question marks are printed for any parts of the date that have illegal values.

xpr_time_ay

Takes standard ProDOS time word in AY and prints time in the form hh:mm xM. If AY=0, prints blanks instead.

xProDOS_err

Prints ProDOS error message from A and aborts to the command line. This routine closes any files you opened (provided you didn't fiddle with LEVEL) and cleans up the stack. You generally don't have to worry about cleaning things up.

Input and output redirection are cancelled, but print spooling is not disturbed.

xProDOS_er

Prints ProDOS error without a bell and returns (does NOT abort).

xerr

Aborts to Davex's command-line prompt. Use this routine if you print an error message (with xmess) and want to abort like xProDOS_err would. (Note that the error message may be redirected to a file or printer. You may want to call xredirect with A=1 to suspend I/O redirection before calling xmess to print the error.)

xpush_level

Prepares to open a new directory level; must be called before dir_setup is called.

xdir_setup

Opens a new directory level. Call xpush_level first. Use xreadldir to read entries from the directory. Call xdir_finish when there are no more entries. On entry to xdir_setup, A and Y should point to a complete pathname or a partial pathname RELEATIVE TO THE PREFIX (compare xdir_setup2).

xdir_setup2 [Davex v1.23+]

Just like xdir_setup, but the pathname pointed to by AY should be either (1) complete or (2) partial RELEATIVE TO THE DIRECTORY ALREADY OPEN. This is useful for commands that traverse a subdirectory structure, since you can just call this routine with the directory name.

xdir_finish

Closes the current directory level and re-opens the previous one, if one was open. To exit normally, you must call this routine once for each call to

xdir_setup you make. (If you jump to xerr or xProDOS_err, don't worry about it.)

xreadldir

Reads one entry from the current directory, opened with xdir_setup. Returns SEC if there were no more entries in the current directory. The directory entry is stored at "catbuff".

xpoll_io

Should be called while waiting for keyboard input. This gives Davex a chance, for example, to send data from spooled files to the printer. (This routine is called automatically during xcheck_wait, xrdkey, and xgetln calls.) A, X, Y, and P are PRESERVED. Also, xpoll_io increments the two-byte random number on zero-page (\$4E and \$4F).

xmmgr

A crude memory manager--allows external commands to use the space between the end of Davex and the beginning of the external command. (Assemble external commands to end as close to \$B000 as possible to maximize this free space.)

Input in X:

mli_close	free all dynamic memory
mli_open	alloc A pages from low mem; SEC=out of mem; return A=1st page
mli_read	return number of free pages in A, Y=0
mli_gfinfo	return lowest free page number into A, Y=0
mli_write	set highest available page to A

xpmgr

PathManager: performs common operations on pathnames. The format of a call is:

```
jsr xpmgr
dfb COMMAND
dw PARM1 [,PARM2]
```

COMMAND is one of the following:

pm_appay
appends path at AY to path at PARM1

pm_appch
appends character in A to path at PARM1

pm_up
removes one segment from end of path at PARM1

pm_slashif
adds '/' to end of path at PARM1 if it doesn't
already end
in '/'

pm_copy
copies length-prefixed path from PARM1 to PARM2
[Davex v1.2+!]

xgetnump [Davex v1.1+]
input: none
output: A=number of parameters given for command
(including all required parameters, even if
they are
0-length strings or pathnames).
X and Y are preserved.

xrdkey [Davex v1.1+]
input: character under cursor (normally use a blank =
\$A0)
output: key pressed (high bit on).

Call this routine instead of using ROM routines for
input. For example, RDKEY, RDCHAR, and GETLN are not
guaranteed to work correctly. Note that input will
may come from an exec file rather than from the
keyboard, unless you use xredirect to suspend I/O
redirection.

xdirty [Davex v1.1+]
No inputs or outputs. Sets a flag to force Davex to
try to re-save the %config information. (The
attempted re-saving happens before command prompts.)

xprint_ver
input: version number in A; \$34 prints "v3.4"
(all registers scrambled)

xfman_open
input: AY points to pathname of a file
output: CLC, A = file reference number
SEC, A = ProDOS error code

xfman_open and xfman_read provide a way to read text
and AppleWorks Word Processor (AWP) files without
caring which kind of file is which. The resulting
stream of data

Additional filetypes may be interpreted in the future.
In general, these routines perform a reasonable
mapping from some non-text files into a legible
sequentially-readable format.

Warning: Do not attempt to open more than one file at
a time using xfman_open. It is not currently
supported.

xfman_read

input: A=reference number returned from xfman_open
output: CLC, A = character
SEC, A = ProDOS error code

Returns the next character from a file opened with
xfman_open.

No special way is provided to close a file opened with
xfman_open; close with a ProDOS call if necessary.

This may be inadequate if these file manager routines
are ever enhanced to deal with more than one file open
at a time.

xshell_info [Davex v1.25+]

input: X=request code
output: CLC, requested information in registers/etc.
SEC, requested information not available

X=0: Get Davex version in A, Y.

For version \$a.bc, A=\$ab and Y=\$0c.

X=1: Get alias buffer (AY=address, X=size in pages)

X=2: Get history buffer (AY=address, X=size in pages)

X=3: Get internal filetype table (AY=address)

X=4: Get internal filetype name table (AY=address)

--notes--

External commands should not open any files below stdlevel,
which is the current ProDOS file level when the shell
executes an XC. Davex automatically closes any files open at
or above stdlevel when the external command finishes. Davex
guarantees that an XC will be able to open 3 files, but it
does not guarantee any more than that (Davex may have up to 5
files open already [spooling, exec, output redirection to
disk, wildcard expansion and maybe one more in the future];
the ProDOS limit is 8 open files).

When an external command gets control at its execution
address, Davex has just finished calling xgetparm_n for
parameter number 0. So, for example, a command whose first
parameter is a pathname may start out by storing A and Y into
a ProDOS parameter block, since AY will have the address of
the value of the first parameter.