Enhancing XEDIT as an APL Editor

Rexford H. Swain

Independent Consultant
8 South Street
Washington, CT 06793
(203) 868-0131
(212) 242-5816

ABSTRACT

While APL2 provides a built-in editor, under VM it also offers a convenient interface to XEDIT, the VM system editor. XEDIT is an incredibly powerful general-purpose editor, well worth mastering for its many uses outside the APL environment. But it is missing some facilities specifically tailored for APL function editing. This paper describes a solution to one specific deficiency: the lack of a "name" or "whole word" change command. The XEDIT macro presented here, CN (for "change name"), corrects the problem and also offers an introduction to REXX, IBM's SAA command language, in a context familiar to APL'ers.

WHICH EDITOR?

The APL2 ") EDITOR" command allows each user to select the editor that will be used as the "del" editor for editing functions (and variables). In the VM environment, the two significant choices are the native fullscreen editor (Editor 2) and the VM system editor (XEDIT). Since programmers spend a tremendous amount of time using an editor, the choice is important. And since each editor takes time to learn, its power and range of applications must be carefully considered.

Editor 2 provides features specific to APL but is only available within APL. XEDIT is a far more robust general-purpose editor. It's also available throughout the VM system, so all skills learned may be leveraged across the full spectrum of VM applications. Since most mainframe APL applications now interact with other external facilities, non-APL skills are becoming essential for the APL developer.

XEDIT allows the use of the VM system interpreter, REXX, as its macro language. REXX is a command language with all the power of a full-fledged programming language, so there is virtually no limit to the capability of XEDIT macros. In fact, entire applications can be developed using REXX with XEDIT as a screen manager. IBM has included REXX in its System Application Architecture (SAA) specification, so any time invested in mastering this language will be rewarded not only elsewhere in VM but in other environments as well.

Unfortunately, there are a few weaknesses in XEDIT from the point of view of an APL programmer. Presented here is a solution for one of those weaknesses: the lack of a "name" (sometimes called "whole word" or "token") change command.

WHOLE WORD CHANGES

The idea is to avoid accidentally changing one string to another in an inappropriate context. For example, when editing a document, we may want to change "data" to "fact", but we would not want to change "database" to "factbase". PC word processing programs usually call this filtering of matches a "whole word" change feature. It is not sufficient to wrap spaces around the strings -- changing " data " to " fact " would change whole words when they occurred in the middle of a sentence, but would overlook occurrences at the beginning of a line and when abutting punctuation, such as at the end of a sentence.

When editing APL programs, this feature is even more useful. The idea of a "word" must be augmented to include any APL token (identifier name or numeric constant). The following are examples of desirable behavior:

Change	To	In	But Not In
data	fact	'the data'	'database'
DATA=	FACT<	+/DATA=0	MYDATA=0
I	J	I+1	IF
+19	+21	(AGE+19)	YEAR+1900

With Editor 2, these changes can be made by adding an "N" (for "name") as a suffix to the change command (e.g. "[/old/new/N]"). XEDIT does not directly support such changes, but the CN macro presented here does.

The technique is to locate a raw match and then check the edge characters. If the first character of the string is possibly part of a name, then the preceding character should not be; and if the last character of the string is possibly part of a word, then the following character should not be.

For example, suppose we want to verify an occurrence of the string "DATA=". Because the first character of the string is a letter (part of a name), the preceding character must not be -- "...+/DATA=..." would satisfy the rule, but not "...MYDATA=...". Since the string ends with a symbol (which cannot possibly be part of a name), we don't care what the following character is -- either "...DATA=+/B..." or "...DATA=B..." would qualify.

SYNTAX

CN's behavior is exactly the same as XEDIT's native CHANGE command except that matches are not changed if they begin or end in the middle of a token. CN's syntax, like CHANGE's, is:

```
CN/old/new/target p q
```

Where...

`CN`	Name of the macro.
`/`	Delimiter. Any character not occurring in `old` or `new` may be used.
`old`	String to be changed. If empty, the `new` string is inserted at the beginning of lines.
`new`	Replacement string. If empty, the `old` string is deleted.
`target`	Specifies the end of line range to be searched (range begins at current line). Default is `1`.
`p`	Number of occurrences per line to change, or "`*`" meaning all. Default is `1`.
`q`	Relative number of first occurrence to change on each line. Default is `1`.

The target can be one of...

Absolute line number:

```
:5
```

Relative displacement from current line:

-5	above current line
5	plus assumed
+5	below current line

File line range:

-*	top of file
*	plus assumed
+*	end of file

Line name:

```
.foo
```

String expression:

`/foo/`	simple
`/a/&¬/b/`	complex

The search scope is from the current line to, but not including, the line specified by the target.

MACRO HIGHLIGHTS

REXX code is remarkably easy to read. Comments begin with "`/*`" and end with "`*/`". There is no assignment arrow, but variable assignments with the more traditional "=" are unmistakable. And only APL'ers have any difficulty recognizing "`*`" and "`/`" as multiplication and division. Logical and is "`&`"; or is "`|`"; not is "`¬`". Catenation may be performed with "`||`" or simple abuttal. An array name and subscript are separated by a period.

The control structures, such as "`if`...`then`...`else`...", are easily understood. The "`do while`..." instruction uses a leading test; "`do until`..." defers the test until the end of the block. The "`do forever`" construct is terminated by a "`leave`" instruction.

Most statements are executed by the REXX interpreter. Quoted strings are taken literally (not subject to variable resolution). Strings not recognized by REXX (such as "`'COMMAND LOCATE' target`") are directed to the calling environment (XEDIT). Other macros may be called as subroutines ("`'MACRO PARSE` ...'").

As in APL, logical tests return boolean values. But there is little usage distinction between datatypes. Values may be used in either a numeric or character context and are converted accordingly.

Order of execution takes some getting used to for an APL'er, but is well documented and usually works in a seemingly natural way. Evaluation is left to right, with the following precedence: prefixes, exponentiation, multiplication and division, addition and subtraction, concatenation, comparisons, and, or. [Sigh; memorization is hopeless -- a crib sheet is the only solution.] Parentheses may be used, or course, to force (and/or clarify) order of execution.

A variety of built-in functions are available. Arguments (sometimes more than two!) are passed within parentheses, separated by commas. The functions "`push`" and "`pull`" write and read from a stack; here they are used to communicate with the "`PARSE`" macro, an IBM-supplied utility which helps to parse CN's argument. The "`parse`" instruction is a remarkably useful tool which separates strings into several values. The function "`substr`" indexes sub-strings from strings, "`strip`" removes blanks, "`index`" is like dyadic iota (but returns zero when not found), "`length`" is like shape, "`delstr`" deletes sub-strings, "`left`" is like positive take, "`right`" is like negative take, and "`x2c`" converts hex values to characters. There are about 60 more REXX functions not used in CN.

XEDIT, like APL, allows the programmer to query and change almost every aspect of its execution environment with the "`EXTRACT`" and "`SET`" commands. (Variables such as "`line.1`" and "`target.2`" are set by "`EXTRACT`".)

CONCLUSION

The CN macro does not, of course, make up for all of XEDIT's shortcomings as an APL editor. But since each of us wants different things from an editor, a good editor must be customizable, and CN demonstrates that XEDIT may be enhanced to suit our various needs.

I hope this glimpse of the potential of XEDIT and REXX will serve as an enticement to learn and use tools outside the APL environment.

APL89 SOFTWARE EXCHANGE

CN and a few more related name locate/change macros are available via the APL89 software exchange:

Macro	Like IBM's	Purpose
`CN`	`CHANGE`	Change Name.
`LN`	`LOCATE`	Locate Name.
`CLN`	`CLOCATE`	Column Locate Name.
`ALLN`	`ALL`	Uses `LN` rather than `LOCATE`.
`SCHANGE`	`SCHANGE`	Accepts `LN` or `CLN`, in addition to `LOCATE` or `CLOCATE`, on command line.

Each macro contains comments explaining its function.

ACKNOWLEDGEMENTS

I am indebted to Bob Hendricks for revealing the full depth of XEDIT and for patiently answering numerous questions about REXX.

Thanks also to Roy Sykes for a utility function named "`SESEARCH`" (for "syntactic element search") on the Scientific Time Sharing Corporation system (circa 1974) which was my first exposure to this concept in its APL context.

REFERENCES

APL2 Language Reference; IBM Order Number SH20-9227. (See chapter 8 for information on editors.)

VM System Product Interpreter User's Guide; IBM Order Number SC24-5238.

VM System Product Interpreter Reference; IBM Order Number SC24-5239.

VM System Product Editor User's Guide; IBM Order Number SC24-5220.

VM System Product Editor Command and Macro Reference; IBM Order Number SC24-5221.

SEE ALSO

Weintraub, David M.; *APL2 and the CMS System: Exploiting the APL2/REXX Connection*; APL88 Conference Proceedings.

Brenner, Norman; *Editing APL Objects With CMS XEDIT*; APL84 Conference Proceedings.

```
/* ---------------------------------------------------------- */
/* CN XEDIT: like XEDIT's Change command but won't break Names */
/* Syntax is same as change command: CN/old/new/target p q     */
/* Effect similar to APL2 Editor 2's change command "N" suffix */
/* E.g., CN/DATA/FACT/ will change "+DATA=" but not "+MYDATA=" */
/* Also good for "whole word only" changes in text files       */
/* Rex Swain, New York City, 24 January 1989                   */
/* ---------------------------------------------------------- */

 parse arg args                               /* all arguments */
 if args = '' then                            /* empty?        */
    exit errmsg(5,'Missing operands')         /* scold user    */

/* ----- Use IBM's PARSE macro to help parse argument -------- */

 push args
'MACRO PARSE 1 Dblstring Target Line'   /* see HELP XEDIT PARSE */
 if rc > 1 then
    exit errmsg(rc,'Invalid operand')
 pull n

 pull d                                        /* dblstring   */
 parse var d d.1 d.2 d.3 d.4 d.5 d.6
 del = substr(args,d.1,1)                      /* delimiter   */
 old = substr(args,max(1,d.3),d.4)             /* old string  */
 new = substr(args,max(1,d.5),d.6)             /* new string  */

 if n >= 2 then do
    pull t                                     /* target      */
    parse var t t.1 t.2
    target = substr(args,t.1,t.2)              /* target      */
    end
 else
    target = ''

 if n >= 3 then do
    pull l                                     /* line        */
    parse var l l.1 l.2
    pqx = substr(args,l.1,l.2)                 /* p and q     */
    end
 else
    pqx = ''

/* ----- Parse and verify p and q --------------------------- */

 pqx = strip(pqx)
 if '-' = target ,            /* change command allows "- *" */
  & '*' = left(pqx,1) then do /* as a target, but PARSE sees */
    target = '-*'             /* just "-"; so shift "*" from */
    pqx = strip(substr(pqx,2)) /* pqx into target            */
    end
 if '*' = left(pqx,1) then    /* change command allows "*1"; */
    pqx = '*' substr(pqx,2)   /* force "* 1" so it'll parse  */
 parse var pqx p q x
```

```
 if target = '' then target = '1'  /* default is current line  */
 if p      = '' then p      = '1'  /* default is once per line */
 if q      = '' then q      = '1'  /* default is first occurr  */

 if p ¬= '*' & ¬ datatype(p,'Whole') then
    exit errmsg(5,'Invalid occurrences per line:' p)

 if old == '' & ( p > 1 | p = '*' ) then
    exit errmsg(5,'Invalid occurrences per line' ,
            '(when old string empty):' p)

 if ¬ datatype(q,'Whole') then
    exit errmsg(5,'Invalid relative first occurrence:' q)

 if x ¬= '' then
    exit errmsg(5,'Superfluous operand:' strip(x))

 if old == '' & new == '' then exit   /* change exits quietly  */

/* ----- Establish local environment; verify target ---------- */

'COMMAND EXTRACT /LINE/COLUMN/WRAP/MSGMODE/STAY/CASE' || ,
                '/ALT/ZONE/SPAN/VARBLANK/STREAM'

 origline = line.1                    /* drop bread crumb      */
'COMMAND SET MSGMODE OFF'             /* quiet                 */
'COMMAND SET WRAP OFF'                /* targets don't wrap    */
'COMMAND LOCATE' target               /* try finding target    */
 if rc = 2 then
    exit targex(rc,'Target not found:' target)
 if rc = 5 then
    exit targex(rc,'Invalid target:' target)
 tofeof = rc = 1                      /* TOF or EOF reached?   */
'COMMAND EXTRACT /LINE'               /* where are we now?     */
 if line.1 >= origline then do        /* target below original */
    targline = line.1 - 1             /* adjust up             */
    topline  = origline               /* first line to search  */
    botline  = targline               /* last line to search   */
    end
 else do                              /* target above original */
    targline = line.1 + 1             /* adjust down           */
    topline  = targline               /* first line to search  */
    botline  = origline               /* last line to search   */
    end
'COMMAND  LOCATE :'topline            /* start at higher line  */
'COMMAND CLOCATE :0'                  /* be sure to hit col 1  */

 changed  = 0                         /* count occurrences...  */
 lines    = 0                         /* ...and lines changed  */
 prevline = 0                         /* force ¬= below        */

 alph = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' || ,
        'abcdefghijklmnopqrstuvwxyz' || ,
        '0123456789_'                || ,
    x2c('41 42 43 44 45 46 47 48 49'    ,
        '51 52 53 54 55 56 57 58 59'    ,
        '62 63 64 65 66 67 68 69 BB FC 90 A0')

/* To detect APL names, the three lines above add underscored  */
/* A to Z, delta, underscored delta, quad, and high minus.      */
/* Specified in hex to facilitate up/downloading of this file.  */

 clarg = del||old||del             /* clocate argument      */
 charg = del||old||del||new||del   /* change argument       */

 lnew  = length(new) - 1           /* advance after change  */

 first = 0 = index(alph,left(old,1))  /* first char not alph?  */
 last  = 0 = index(alph,right(old,1)) /* last char not alph?   */

'COMMAND SET CASE' case.1 'RESPECT' /* change has respect... */
'COMMAND SET SPAN OFF'              /*  does not span        */
'COMMAND SET VARBLANK OFF'          /*  ignores varblank on  */
'COMMAND SET STREAM ON'             /*  ignores stream off   */

 if p = '*' then limit = 999999999  /* change all on a line  */
 else            limit = p + q      /* last occurr to change */

/* ----- Begin locate/examine/change loop -------------------- */

 do forever                          /* well, until leave     */

   'COMMAND CLOCATE' clarg           /* raw clocate           */
    if rc ¬= 0 then leave            /* rc=2 means not found  */
   'COMMAND EXTRACT /TARGET/CURLINE' /* where is hit?         */
    if target.1 > botline then leave /* beyond target?        */

/* ----- Does hit qualify as a "name"? ----------------------- */

    text  = curline.3                /* text of current line  */
    left  = target.2                 /* index of first char   */
    right = target.4                 /* index of last char    */
    prev  = substr(' 'text,left,1)   /* char before old       */
    next  = substr(text,right+1,1)   /* char after old        */

    name = ( first | 0=index(alph,prev) ) ,
         & ( last  | 0=index(alph,next) )

/* ----- Change if a name and count within limits ------------ */

    if name then do

       if prevline ¬= target.1 then do      /* hit new line?  */
          prevline = target.1               /* swap           */
          hits = 0                          /* restart count  */
          end
       hits = hits + 1                      /* hits this line */
       if hits >= q & hits < limit then do  /* within limits? */
         'COMMAND SET ZONE' target.2 zone.2 /* restrict zone  */
         'COMMAND CHANGE' charg             /* change it!     */
         'COMMAND SET ZONE' zone.1 zone.2   /* restore zone   */
         'COMMAND CLOCATE :'left + lnew     /* skip over new  */
```

```
          changed = changed + 1             /* hits changed   */
          lines = lines + ( hits = q )      /* lines changed  */
          end

       end                                  /* end if syn     */

    end                                     /* end do forever */

/* ----- End loop; exit with status message ------------------ */

 if changed = 0 then exit restex(4,'No lines changed')
'COMMAND SET ALT' 1+alt.1 1+alt.2
 exit restex(0,,changed left('occurrences',11-(changed=1)) ,
             'changed on' lines left('lines',5-(lines=1)) )

/* ----- restex: Restore environment prior to exit ----------- */

restex: parse arg xrc,emsg,msg

'COMMAND SET WRAP     ' wrap.1              /* restore...     */
'COMMAND SET MSGMODE  ' msgmode.1
'COMMAND SET STAY     ' stay.1
'COMMAND SET CASE     ' case.1 case.2
'COMMAND SET SPAN     ' span.1
'COMMAND SET VARBLANK ' varblank.1
'COMMAND SET STREAM   ' stream.1            /* ...environment */

'COMMAND CLOCATE :'column.1            /* always restore col  */
 if stay.1 = 'ON' then                 /* if stay on          */
   'COMMAND LOCATE :'origline          /* original line       */
 else                                  /* if stay off         */
    if tofeof then                     /* last line examined: */
      'COMMAND LOCATE' target          /* target itself...    */
    else                               /* ...or...            */
      'COMMAND LOCATE :'targline       /* ...target +|- 1     */

 if emsg ¬= '' then return errmsg(xrc,emsg)  /* error message */
'COMMAND MSG' msg                            /* normal message */
 return xrc

/* ----- targex: Target problem; prepare to exit ------------- */

targex: parse arg xrc,emsg

'COMMAND SET WRAP    ' wrap.1          /* restore...          */
'COMMAND SET MSGMODE' msgmode.1        /* ...environment      */
 return errmsg(xrc,emsg)               /* return rc for exit  */

/* ----- errmsg: Display error message and original command -- */

errmsg: parse arg xrc,emsg

'COMMAND EMSG' emsg                 /* display error message    */
 parse source . . . . . cmd .       /* name/syn of this macro   */
'COMMAND CMSG' cmd args             /* redisplay with arguments */
 return xrc                         /* return rc for exit       */
```