copied from [illegible]

RC 8352

P-EDIT

A Text Editor for Parametric Files

USER GUIDE

July 1980

Vincent Kruskal

P-EDIT is a new text editor that has been developed to provide a more convenient way to create and maintain CMS files for which there can be many different versions. This document is intended as a reference manual for P-EDIT users. It is also available on-line while P-EDIT is being used.

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

RC8352 (#36326) 7-10-80
Computer Science 133 pages

Table of Contents

Introduction

For the most part, this document is automatically generated from the same files used by the on-line documentation facility of P-EDIT. Also included is a reprint of an article, previously published, that describes how a text editor, such as P-EDIT, can provide a much better tool for dealing with different versions of a file by allowing all the versions to be edited together as one object.

It was this need that made the implementation of still yet another editor necessary. P-EDIT is a joint project of the Automatic Programming and the System Laboratory Departments at IBM Research. It was created by myself and Paul Kosinski. Peter Sheridan provided the Boolean expression simplifier that has been described in "A Formula Decision/Simplification Program", IBM Research Report RC7132, May 18, 1978. Leroy Junker has been an invaluable early user by being extremely communicative and inquisitive.

P-EDIT currently has a small body of users who have been attracted to it for a number of reasons:

> The ability to edit the many versions of a file as one object called a *parametric file*. Anyone who is currently using UPDATE, the PL/I Preprocessor, or simply contending with lots of similar files, should consider this new way of dealing with such situations.
>
> The ability to remember the effects of previous commands so that disasters can be easily corrected. The UNDO command can be used to remove all effects of any number of commands, while the UNGO command can be used to remove just the positional effects of any number of lines (the effects of redefining which line is the one being edited).
>
> The ability to switch easily back and forth between all the files being concurrently edited. Lines can be given names, called *tags*, and by using the GO command any of them can become the line being edited. In fact, most of the P-EDIT commands that operate over a specified range of lines permit tags, as well as line counts, to be used for that specification.
>
> The ability to edit files whose lines are up to 65,535 characters wide.
>
> Extensive compatibility with existing text editors. All the common commands will be familiar to the experienced user (NEXT, UP, DELETE and such). .
>
> An interface to EXEC2 that permits easy extention of the available P-EDIT

facilities. A large library of such extensions is publicly available, adding such additional facilities as:

- extensive on-line documentation
- indentation of structured files
- spelling error detection with suggested changes
- entering P-EDIT commands while in INPUT mode
- generating lists of CMS files, CP spool files and files being edited
- operations on those files without having to retype the information
- knowledge of how files of many different types should be edited

The documentation that follows has extensive cross-references. The root of this network of cross-references is in the description of the TELL macro in the chapter on macros, a good place to start for someone unfamiliar with P-EDIT. Within each chapter, the topics are arranged alphabetically. When a topic has more than one name, it appears under one of those names and the others refer to it.

A New Technique for the Development and Maintenance of Parametric Programs

Vincent Kruskal

Computer Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

This paper was originally published in the *Proceedings of the Seventeenth Annual Technical Symposium of the ACM (Tools for Improved Computing in the 80's)*, June 15, 1978, prior to the implementation of P-EDIT. The version here has been updated to reflect the actual implementation and the vocabulary that eventually evolved.

Abstract

A *parametric program* is a model of a group of related programs which contains the logic necessary for each of those programs to be realized, based on the setting of a number of parameters. This concept of parametric program is merely a generalization of a whole slew of techniques that have been used in the past to represent, in one module of code, different versions of a program, the history of changes to a program, and the like.

Previous techniques for representing parametric programs have typically involved explicit coding of the necessary control information intermixed throughout the program text. Such control information is usually in the form of compile-time condition statements which test Boolean expressions of parameter values. Such parametric programs can easily become unreadable. The reader must look at a jungle of control information and fragments of program text and try to imagine what some particular version of the program might look like.

What is needed is a high-level way of dealing with a parametric program which hides all this complexity from the user and lets him return to the world of dealing with a straight-forward program. A simple and elegant method is described for automating the bookkeeping needed to develop and maintain parametric programs. Ways are discussed for naturally grafting this method onto a display oriented text editor.

The earliest computer programs were small and were coded by same person interested in using them. As programs became more complex, teams of programmers were needed to code them. As these teams became larger, they became more professional and more removed from the user. To justify the expense of these programming teams, the programs had to be more general-purpose to support a larger body of users. This phenomenon made necessary the invention of numerous automatic bookkeeping aids to keep track of the additional complexity:

Update utilities permitted a set of related modifications of a program to be packaged as a single update deck. These update decks, when applied to the original program in chronological order, would produce the program's current version. This facility was originally created simply to permit the modification of large programs stored on tape. However, it was soon discovered that it was better to keep the original program and the update decks as the official representation of the program's current version and to discard the actual current version after compilation. This gave a representation of the program which included the history of changes.

Program development systems permitted the members of a programming team to share a common library of code. While a large program is under development, the various programmers will be out of sync with one another to some degree. When the definition of a module of code changes, all the programmers can't instantly accommodate themselves to the new version. So at any time, each programmer has his own view of the current state of the program under development. If each programmer had his own copy of the program, he would be spending most of his time installing fixes to his copies of other programmers' modules. So program development systems were invented which allowed the sharing of a common library of code. These systems know the management structure of the programming team, enforce ownership rules, and provide the appropriate view of the program to each programmer.

Customizers permitted programs to be automatically tailored according to a questionnaire filled out by the end-user. Thus, programs could be more general-purpose than any single user would want while remaining efficient. This was far better than simply giving the user a general-purpose program which would contain large amounts of unneeded code, cause unnecessary testing of options during execution, and be much harder for him to modify. In addition, most programming languages have some aspect which cannot be specified during execution, declaration of variables is a typical example. Such things can be made optional only by using a customizer.

Although not generally recognized, all these bookkeeping aids have one thing in common: a representation of programs that permits them to contain alternative sequences of code complete with the necessary control information to define when each sequence is appropriate. Any program with such a representation is called here a *parametric program*. A parametric program contains all the information needed to determine a particular program, once the values of the parameters are given. When all the parameters have been given values, the resulting program is called an *instantiation* or, sometimes, a *version* of the parametric program. When some, but not all, of the parameters have been given values, we sometimes speak of the *partial instantiation* defined by those which have. Clearly, a partial instantiation appears to be, itself, a parametric program. However, there is a subtle difference that will be discussed latter.

Using this concept of parametric program, one could think of update utilities as having a single parameter to control which update deck should be the last one applied. Likewise, program management systems have, most simply, one parameter per programmer; each programmer's view of the program under development would be defined by a value for each of these parameters telling how current he is relative to each of the other programmers. And, of course, customizers have one parameter per question on the questionnaire, each with one value for each possible answer.

Each of these applications of the concept of parametric program traditionally had its own representation and techniques for dealing with it. Each of these techniques was inappropriate for the other applications. This paper describes a tool for manipulating parametric programs in a uniform and simple fashion: a single tool that can satisfy all these needs simultaneously while making each easier for the programmer.

Customizers

The most complicated parametric programs are those written for customizers. It is in this application that we find large numbers of independently specified parameters interfering with each other by controlling the same sequences of code. Before going further, we should look at the best technique used in the past to produce parametric programs for customizers, for this was the only technique that could have possibly satisfied the desire for a uniform way of manipulating parametric programs.

Early compilers, including assemblers, merely provided the facility of translating from a human-oriented language to machine language. But because of the need for customizers, these compilers soon provided additional facilities for modifying the program during compilation. These facilities included compilation-time variables and control statements. At the beginning of each program would be a section where these compilation-time variables would be assigned values. Comments would be liberally used in this section to explain the possible values for each compile-time variable and the effect of each value. Throughout the remainder of the program would be compile-time control statements so that the compiler would ignore all the text which was inappropriate for the particular version defined by the compile-time variables. For example, the preprocessor of IBM's PL/I compiler permits a subset of PL/I, preceded by a percent-sign, to control compilation. Fragments of a PL/I parametric program follow:

```
PROG: PROCEDURE() OPTIONS(MAIN);
                            ...
                            ...
  %VERSION = 5.6;                               /* VERSION DESIRED */
  %SYSTEM = 'VM';                               /* 'MVS' OR 'VM'   */
                            ...
                            ...
  I = I + 1;
  %IF (VERSION >= 4.0) & (SYSTEM = 'VM') THEN CALL ADJUST(I);
                            ...
                            ...
  END;
```

This kind of facility is clearly arbitrarily powerful in theory. However, as a practical matter, it will get out of hand when the parameterization of the program becomes complex. When a great number of parameters control the same sequence of code, the Boolean expressions get complicated and hard to understand. When the sequences of code get small because a great deal of control is being exercised, it becomes impossible to look at the program and get any idea of what an instantiation might look like. Because of this, customizers have either permitted the user only modest options or have been too expensive.

It should now be clear why this technique for handling parametric programs has not been able to serve all the needs: it had trouble serving just the needs for customizers. If in addition to implementing user options, the entire history of the program and all the various versions under development were encoded by explicit control statements, visible to anyone reading the program, nobody would ever understand it.

The Editor

The basic idea here is to change the program editor that the programmer already is using so that it can be told the programmer's intent in making any modification to his program. For concreteness, let us assume that we are talking about one of the many text editors that have evolved from typewriter context editors to use display terminals. These editors are built around the concept of a *cursor* which is always pointing at some line of the text called the *current line*. This cursor can be moved relatively, absolutely, or contextually. For example:

`UP 5`	Move cursor up five lines.
`LINE 1057`	Move cursor to line number 1057.
`LOCATE "CALL"`	Move cursor to the next line containing a CALL statement.

Once at the desired location in the text of the program, either the current line can be modified or new lines can be inserted. For example:

`CHANGE "CSLL"CALL"`	Correct the spelling of CALL.
`REPLACE CALL SUBR(X)`	Replace the current line.
`INSERT Y = Y + 1`	Add a new line after the current one.

It should be noted that these editors know nothing about the language the programmer is using: they merely have the concept of character and line. Thus they are often used to produce English documents as well as programs and the techniques described in this paper apply equally well to parametric documents as they do to parametric programs. Indeed, being able to maintain the documentation of a parametric program using the same system is necessary for a well functioning development tool.

The display terminal is often used merely to display a window of the text being edited around the current line. We will assume that the terminal has some method of highlighting lines of text which we will call *bright*.

Building Parametric Programs by Incremental Programming

Incremental programming is my name for making a small, single-purpose modification to an existing program. This kind of programming involves the least complexity of any and is the most common assignment for a novice programmer. He has the system organization and data structures rigorously defined for him and need only hold in his mind the simple change he wishes to make.

If the task of creating a parametric program from a simple program can be reduced to a sequence of incremental programming steps, we will have accomplished, in that sense, the complete automation of the complex bookkeeping that is necessary in dealing with a parametric program.

In order to build a parametric program in this fashion the programmer must start with a *base program.* This base program is simply a non-parametric program built and debugged in the traditional way using the editor. If he is creating a customizer, the base program is one possible instantiation of the program which he feels is typical. If the purpose of the parametric program is to record changes to the program, the base program is the original version.

Once the base program is completed, all further uses of the editor will use the new parametric features which have been grafted onto the editor's original programmer interface. Prior to making any change to his program, the programmer would inform the editor of his intention by specifying a parameter name and value(s). The following examples are typical for the application mentioned:

`BILLING.METHOD = PRE.BILLING`	Accounts receivable customizer
`BUG564 = FIXED`	Correction maintenance system
`VERSION >= 5.6`	Release maintenance system
`KRUSKAL >= 5.6`	Program development system

From this point on the programmer would proceed to use the editor in a completely normal fashion. The editor would monitor every change to the program and automatically build a data structure which associates the new text with the programmer's intention and associates the old text with the negative of his intention (example: BUG564 ¬= FIXED). Thus the program would take a form internally such as:

```
DO WHILE (Y > Z);                                          TRUE
    CALL SUBR(X);                                          BUG564 = FIXED
    CSLL SUBR(X);                                          BUG564 ¬= FIXED
    END;                                                   TRUE
```

Note that each line internally has two fields: the program text and a Boolean expression. If a line is in all instantiations of the parametric program, its Boolean expression is TRUE. If not, its Boolean expression would be something more constraining. There is no notion that two lines are alternatives to one another built into the editor; each line is either in an instantiation or not. If two lines are, in fact, alternatives, they would be consecutive and have Boolean expressions that are mutually exclusive.

In the base program, all Boolean expressions are TRUE. Each time the programmer informs the editor of a different intention and modifies the program, the Boolean expressions become evermore restrictive. Because the editor always retains the text associated with the negative of each intention, there is always a partial instantiation corresponding to each value of a parameter that has been mentioned plus one that corresponds to the case where that parameter has none of those values. Thus the editor has no notion of a limited range of values for each parameter and no declaration of possible parameter names or the values they might take is needed (although one might be imposed so that typing errors might be detected).

It was mentioned earlier that while partial instantiations bare a remarkable similarity to parametric programs, there is a subtle difference between them. That difference is lies in the need to restrict the range of parameter values. As shown above, parametric programs have no such need. A partial instantiation, however, cann't exist under an assumption that contradicts the assumption that induced it. Thus the partial instantiation induced by BUG564 = FIXED must exclude from the range of possible values for BUG564 any value other than FIXED.

```
Boolean-expression      IS  Boolean-term
                        OR  conjunction
                        OR  disjunction
                        OR  function
Boolean-term            IS  constant
                        OR  relational-expression
                        OR  (Boolean-expression)
                        OR  ¬(Boolean-expression)
conjunction             IS  Boolean-term & Boolean-term
                        OR  Boolean-term & conjunction
disjunction             IS  Boolean-term | Boolean-term
                        OR  Boolean-term | disjunction
relational-expression   IS  name relation-value
                        OR  value L-relation-name L-relation-value
relation                IS  E-relation
                        OR  L-relation
                        OR  G-relation
E-relation              IS  =
                        OR  ¬=
L-relation              IS  <
                        OR  <=
G-relation              IS  >
                        OR  >=
constant                IS  TRUE
                        OR  FALSE
value                   IS  base-36-signed-Dewey-decimal-number
                        OR  function
name                    IS  letter
                        OR  letter base-36-unsigned-Dewey-decimal-number
function                IS  name (arguments)
arguments               IS  argument
                        OR  argument, arguments
argument                IS  value
                        OR
                        OR  (arguments)
```

Boolean Expression Syntax

Boolean Expressions

The term *Boolean expression* is used throughout this paper. Boolean expressions are used to define constraints on parameter values and, therefore, they also define a partial instantiation of the parametric program. Their syntax is in the above figure. Since parameter values are sometimes thought of as character strings and sometimes as Dewey decimal type version numbers, relational operations should be defined to work reasonably on both. When being compared, values are broken up into fields by periods. Each field is a base 36 (digits 0 to Z) number with a possible sign. Fields are compared, left to right (lexically), in the normal Dewey decimal fashion.

The editor needs the ability to compute the AND, OR and NOT of Boolean expressions where these functions return simplified results. In addition, the function, CONSISTENT, is useful for speed since the editor needs it often and it can be made much faster than taking the AND and checking for FALSE.

Masks and Displaying Parametric Programs

Central to this editor for parametric programs is the concept of the *mask*. The mask is a Boolean expression that describes a constraint on the values the parameters can take. The mask is used in two ways. On input, it defines the intent of the programmer's modifications to the parametric program. On output, it describes the partial instantiation the programmer wishes to see. Once the mask is set, there is no way that excluded instantiations can be modified or viewed without resetting it.

Each line of text is said to be *fixed* by the current mask or not. It is fixed if either it is in all instantiations of the parametric program (it has a Boolean expression of TRUE) or, at least, it is in all instantiations that are consistent with the current mask (the mask proves that the Boolean expression on the line is true). A line that is in some, but not all, instantiations that are consistent with the current mask is called *unfixed* (the mask is consistent with both the Boolean expression on the line and the NOT of that Boolean expression). When a line is fixed, it is displayed normally; when a line is unfixed, it is either displayed bright or it is not displayed at all, called *hidden*, and a nearby line is displayed bright.

Only one instantiation can be actually displayed even though, in a logical sense, all the ones that are consistent with the current mask should be displayed, since they are all being edited. So some of the unfixed lines are going to be displayed bright and some are going to be hidden. The choice of which instantiation to display is made incrementally, on an as-needed basis, whenever an unfixed line might be displayed. The first time this decision is made, the editor arbitrarily chooses to display the line. After that, the decision is made to be consistent with all previous decisions in the following manner.

A Boolean expression called the *view control* and a list of Boolean expressions called the *view priority list* are maintained by the editor. Whenever the editor has to decide whether an unfixed line should be displayed, it does so if it is consistent with the view control. If it is also consistent with the NOT of the view control (the decision to display it could have been made, consistently, the other way), the line's Boolean expression is added to the end of the view priority list (lowest priority). Whenever the mask or the view priority list is modified, the view control is recomputed as follows:

> The view control is intialized to the current mask.
>
> Each Boolean expression in the view priority list is examined in turn, starting with the highest priority. If it is consistent with the current value of the view control, it is ANDed to it, thus making it more restrictive.

This operation has the effect of keeping the view control as consistent with prior decisions as is possible, while never contradicting the current mask. It is possible to permit the user to exercise control over the view by allowing him to add to the top of the view priority list.

While this technique guarantees that the view shown the user will be a consistent instantiation, it has the problem that the view chosen might have no lines at some point in the program while other instantiations, consistent with the current mask, do. Thus there is no unfixed line in the view to brighten. When this happens, the previous line is displayed bright even though it is fixed.

The programmer can't see the unfixed lines that are hidden, but they exist as much as the text that he can see in every other sense. If he issues an editor command to move the cursor to the next occurrence of some string, say, and that next occurrence is in such a hidden line, the editor will display a blank line to stand for the hidden line.

As described here and in essence, there is only one mask that controls the editor at any time. However as a convenience for the user, it might be wise to permit a number of masks so that he might more easily exercise control over them. These masks might each have a name or they might be elements of a push-down list (or possibly both). The current mask, as that term has been used here, would be computed by taking the AND of the user's masks.

Handling Interference between Parameters

The real power of this technique does not become apparent until we look to see what happens when different parameters control the same sequence of text. This is the situation where complex Boolean expressions must be formed to control the instantiation process. If the programmer is to continue to operate in the simple incremental programming mode, these complex expressions must be managed automatically by the editor.

Let's take as an example the programmer who is fixing bug number 564. As his first step, he has set the mask to `BUG564 = FIXED`. We want him to focus only on that goal and not even to know about the other parameters. But even when programming incrementally, the programmer can't make modifications blindly. He must look at the existing code and understand it well enough to make sure his change doesn't introduce new bugs. Now most parts of the program needn't be looked at carefully, since it's obvious that his change is independent of them. The programmer need only read the relevant parts of the program. If there are bright lines in those parts, he has been warned that there is hidden code he cannot see.

To handle this case, the editor must permit the programmer to see all the alternatives of a sequence of bright lines and to make his change for each case. Again this must be done in such a way that the programmer always sees a consistent partial instantiation. The SHOW command, issued when the current line is bright, causes the editor to modify the mask so the group of unfixed lines around the current line becomes fixed. This is done by setting the mask to the AND of its previous value and the Boolean expression necessary to fix one alternative of these lines. This, of course, might well cause other lines to become fixed; thus keeping the programmer's view consistent. After the programmer fixes bug 564 for this alternative, he would issue the STEP command. This command would go back to the previous mask and modify it so that another alternative is fixed. When the bug has been fixed for all alternatives, the programmer would issue the UNSHOW command and the mask would be restored.

In order for this operation to act reasonably in the face of possible modifications to the lines whose alternatives are being shown, it is important that the SHOW command precompute all the necessary restrictions to the mask. The STEP command would merely AND each of these, in turn, to the original mask. In addition, for the SHOW operation to act reasonably in the face of possible modifications to the mask prior to the SHOW (possible if named masks, as above, are permitted), it is important to choose the correct restriction to the mask for each alternative (there are many that would act the same as long as the original mask isn't modified). The correct one is that which is least restrictive. This can be computed by choosing any of them and ORing the NOT of the original mask to it.

It should be clear that fixing bug 564 for any alternative might itself involve stepping through the alternatives of another bright line and so forth to any level. Each time this is done the mask becomes more complicated. But the programmer is never doing anything but asking himself "Now how can I fix bug 564 in this code without messing it up?". He has no knowledge of the other parameters; only the unavoidable need to be able to integrate his change into existing code.

Maintaining the Boolean Expressions in the Parametric Program

Whenever a line of text is modified by the programmer, the editor must replace that line in the program with two lines, one with the original text and the other with the new text. It must then associate with the original text the Boolean expression computed by taking the AND of its original Boolean expression and the NOT of the current mask (if this is FALSE, the line is deleted). It associates with the new text the AND of the original Boolean expression and the current mask.

Note that no way has been discussed for a parametric program to become less parametric; after all, bug 564 might have been that two versions are different but should have been the same. To permit this, a MERGE command is needed. This permits consecutive lines with the same text and mutually exclusive Boolean expressions to be combined by ORing the Boolean expressions together and associating the result with one of the lines while discarding the other.

Editor Macro Interface

Most text editors have some sort of macro facility. The simplest kind of macro merely associates a name with a sequence of editor commands. A user who gets weary of issuing the same commands repeatedly can define such a macro and just use its name from that point on. In practice, of course, these sequences of commands should not have to be identical each time. Rather, the user identifies some definable pattern they have in common. So real macro facilities tend to have all the power we expect in programming languages: passing of arguments, conditionals, loop control, invocation of sub-macros and the like. Such a facility makes the editor very extensible with users accumulating and sharing an ever increasing library of macros.

This macro facility is really the hardest to get right when upgrading a text editor to operate on parametric programs. For what is to happen when a macro tests a line of text which is unfixed by the current mask and does something completely different depending on what it finds? Such a test doesn't have a unique outcome: for some parameter values it should have gone one way and for others it should have gone the other. There are really two approaches to this problem: the explicit and the automatic.

The explicit solution is to make all the concepts and control that is available to the user, available to the macros. A command, STATUS, would be provided for a macro to test whether a line is fixed or not. If it is unfixed, the macro would use the SHOW command to step through each alternative. This mimics what the user does when the ambiguity of a bright line makes him uncertain of what editing he wishes to do. It would simply be an error for a macro to test the contents of an unfixed line of text. This would mean, in practice, that all macros must explicitly test the lines of text that it deals with and be prepared to step through each alternative of that text. Clearly this would make the macros more complex. If the editor is already in wide use, we would have the additional problems of rewriting the existing macros and re-educating the users.

However, what these macros have to do to work correctly on parametric programs is sufficiently regular that it could be done for them by the editor automatically. In order for the editor to do this, it must have a facility analogous to a multi-programming supervisor. It needs the ability to introduce a *fork* in the execution of a macro at will. After such a fork, the execution of the macro would split into multiple executions running in parallel, each with a different mask. Were a macro to test a line of text, unfixed by the current mask, and the outcome of that test were different for different alternatives of that line, the editor would introduce a fork in the macro's execution. Each of the resulting executions would use a mask which is the AND of the mask prior to the fork and the Boolean expression which defines one outcome of the test. Let's say, for example, that the current mask is `SYSTEM = MVS` and that the current line and its alternatives are as follows:

```
A = B + C;                                  VERSION < 5.6
A = B + FLOOR(C);                           VERSION = 5.6
A = B + FLOOR(C) + 1;                       VERSION > 5.6
```

If the macro tested the current line to see whether the FLOOR function is called, the execution would fork into two executions. One with the mask (`SYSTEM = MVS & VERSION < 5.6`) and the other with the mask (`SYSTEM = MVS & VERSION >= 5.6`). Each of these executions would proceed independently of each other since they would be using mutually exclusive masks. Any change to the program made by one would be associated with its mask and would be invisible to the other.

This independence of the different executions is crucial, since otherwise the macro would have to concern itself with the parallel execution in the same way that co-operating programs in a multi-programming environment must. But this independence is limited to operations on the parametric program. If the macro facility in the editor permits, as many do, sending of messages to the user, issuing commands to the operating system, or any other communication outside the world of the editor, this independence is shot. The editor must intercept any such communication that occurs after a fork and treat it as an error.

Acknowledgment

This work was done in Patricia Goldberg's Automatic Programming Group at the T.J. Watson Research Center and could not have been done without her encouragement and support. The original inspiration for these techniques came out of discussions I had with Gerry Howe and Irving Wladawsky on the problem of finding a better way of building program customizers. Paul Kosinski and Peter Sheridan are currently working with me to implement a prototype of a text editor for parametric programs, called P-EDIT.

- see DIRECTION

BASIC COMMANDS

MACRO	$	(Execute CMS command)
COMMAND	-	(Execute command in reverse mode)
COMMAND	?	(Print the last few commands)
COMMAND	"	(Execute previous command again)
MACRO	ADD	(Add string to end of line)
MACRO	AGAIN	(Execute remembered command again)
COMMAND	AVOID	(Go to the next line missing specified string)
MACRO	BOOL	(Print Boolean expressions for lines in range)
COMMAND	BOTTOM	(Go to the last line in the file)
COMMAND	CHANGE	(Replace one string with another over range)
RESOLVE	CMC	(Replace one mixed-case string with another over range)
RESOLVE	COPY	(Copy lines from one point to another)
MACRO	CUF	(Replace unique occurence of string in file as specified)
MACRO	CUS	(Replace unique occurence of string on screen)
COMMAND	DELETE	(Delete lines in range)
MACRO	FILE	(Stop editing current file after saving it on disk)
COMMAND	FIND	(Go to the next line that matches pattern)
RESOLVE	FIRST	(Set default FIRST tag for MOVE, etc.)
MACRO	INCHANGE	(Interactive CHANGE)
COMMAND	INPUT	(Enter batch INPUT mode)
COMMAND	INSERT	(Insert a new line after current line)
RESOLVE	LAST	(Set default LAST tag for MOVE, etc.)
COMMAND	LOCATE	(Go to the next line that has specified string)
MACRO	MOVE	(Move lines from one point to another)
COMMAND	NEXT	(Go to the next line closer to EOF)
MACRO	OUT	(Stop editing all saved files)
MACRO	P-EDIT	(Resume editing or read file if not in memory)
COMMAND	PRINT	(Print lines in specified range)
MACRO	QUIT	(Stop editing current file)
MACRO	RANGE	(Set default FIRST or LAST tag for MOVE, etc.)
COMMAND	REPLACE	(Replace current line)
COMMAND	RESUME	(Resume editing a file)
COMMAND	SAVE	(Save current file on disk)
COMMAND	SCROLL	(Go to a line based on display)
MACRO	TELL	(Edit a file to explain specified part of P-EDIT)
RESOLVE	TO	(Set default TO tag for MOVE, etc.)
COMMAND	TOP	(Go to the TOP dummy line)
COMMAND	UNDO	(Undo effect of remembered commands)
COMMAND	UP	(Go to the next line closer to TOP)
COMMAND	X	(Execute or define command abbreviation)

BOOLEAN see BOOLEAN EXPRESSIONS

BOOLEAN EXPRESSION see BOOLEAN EXPRESSIONS

BOOLEAN EXPRESSIONS

The rules for writing Boolean expressions in P-EDIT are very similar to those in PL/I. The principle difference is that parameter names (variables) are always on the left side of relations and parameter values (constants) are always to the right. Therefore there is no need to use quotation marks or such to distinguish names from values. Details follow:

Boolean operators: These consist of '&' for AND, '|' for OR, and '¬' for NOT. Parentheses can be used to group these operations and must be used when mixing AND and OR because there is no built-in hierarchy defined. Example:

```
¬(A=1 & (B=1 | C=1))
```

Constants: The constants, 'TRUE' and 'FALSE', are permitted, but are not often useful.

Relations: Boolean operators act on Boolean expressions, constants, or relations. Relations provided are '<', '<=', '=', '¬=', '>=', and '>'. Relations coded with two characters may not be separated with a blank. Relations take a parameter name on their left and a parameter value on their right. It should be noted that 'P=1' in no sense means that a variable 'P' currently has the value '1'; there are no values in this sense. 'P=1' is significant only in that it contradicts 'P=2', implies 'P>=0', and so forth. A special form is provided for ranges, '0<P<8', where either '<' can also be '<='.

Parameter names: These must begin with a letter and may contain any number of letters, digits and '.'. Any zero digits following a '.' are ignored. Examples:

```
PARAMETER.NAME
NAME5
```

Parameter values: These are a generalization of Dewey decimal notation intended to be adequate for version numbers, integers and symbolic names. A value consists of any number of fields, separated by '.'s. They are compared left to right, lexically (5.6.4 > 5.6). Each field is a base 36 signed integer (digits range from '0' to 'Z'). Thus symbolic values (which would only be compared using '=' or '¬=') act reasonably. Examples:

```
VERSION.1.3.4
CMS
-5.1                    (greater than -5)
7.4.-3
```

Functions: these permit the same kind of extensibility that macros provide for commands. A function may occur wherever a Boolean expression or a parameter value is expected, although a given function would probably make sense only in one context.

Internally and on disk, Boolean expressions must be upper-case and contain no blanks. However, P-EDIT will make the necessary translation when Boolean expressions are read from the terminal or returned by a function (P-FUNC). On disk, Boolean expressions have at least one blank to their left and exactly two blanks on their right (end of the record).

BUILT-IN COMMANDS

COMMAND *	(Do nothing)
COMMAND -	(Execute command in reverse mode)
COMMAND ?	(Print the last few commands)
COMMAND "	(Execute previous command again)
COMMAND ALARM	(Set or print permitting beeping on errors)
COMMAND ALTER	(Replace one character with another over range)
COMMAND APPEND	(Append current file to existing file on disk)
COMMAND ATBASE	(Set base application as defined by current masks)
COMMAND AVOID	(Go to next line missing specified string)
COMMAND BOTTOM	(Go to the last line in the file)
COMMAND BROWSE	(Set or print BROWSE mode which disallows changes)
COMMAND CALLER	(Test who caller was)
COMMAND CASE	(Set or print upper-case folding mode)
COMMAND CHANGE	(Replace one string with another over range)
COMMAND CMS	(Enter CMS subset operation)
COMMAND COLUMN	(Print location of string in current line)
COMMAND COMMAND	(Execute built-in command)

COMMAND COPYFROM	(Copy the range to after the current line)
COMMAND COVERT	(Execute command as part of previous one)
COMMAND DELETE	(Delete lines in range)
COMMAND DELIMSTRING	(Parse a delimited string from a string)
COMMAND DESERIALIZE	(Set or print deserialize mode and columns)
COMMAND DIRECTION	(Test for reverse mode by caller)
COMMAND DROP	(Stop editing a file)
COMMAND EDIT	(Read and edit a file or print files being edited)
COMMAND EDITING	(Print file number of file specified)
COMMAND EDITMODE	(Set or print whether to edit text or Boolean expressions)
COMMAND ERROR	(Act like an illegal command)
COMMAND EXCLUDE	(Restrict mask to exclude range)
COMMAND EXCLUDEVIEW	(Restrict mask to exclude viewed lines in range)
COMMAND EXIT	(Stop execution of READ *)
COMMAND FCOUNT	(Print number of files being edited)
COMMAND FETCH	(Print value of edit variable)
COMMAND FIND	(Go to the next line that matches pattern)

COMMAND FIX	(Restrict mask to fix range)
COMMAND FIXVIEW	(Restrict mask to fix viewed lines in range)
COMMAND FNUMBER	(Print number of file and controlling file)
COMMAND FORMAT	(Set or print format of file (V, F, etc.))
COMMAND FREE	(Print amount of available storage)
COMMAND GETFILE	(Read a file and insert it into current file)
COMMAND GO	(Go to the line named by the specified tag)
COMMAND HEAD	(Set header line of screen)
COMMAND HEADVARS	(Set order of resolution for header line)
COMMAND HILITE	(Set or print method of highlighting lines)
COMMAND IMAGE	(Set or print handling of tab and back-space characters)
COMMAND INPUT	(Enter batch INPUT mode)
COMMAND INSERT	(Insert a new line after current line)
COMMAND LENGTH	(Print length of current line)
COMMAND LEVEL	(Print level of editing)
COMMAND LINES	(Print physical line number and number of lines)
COMMAND LOCATE	(Go to the next line that has specified string)

COMMAND MACRO	(Execute P-EDIT macro)
COMMAND MACROLINE	(Stack calling command)
COMMAND MAKE	(Modify Boolean expressions over the range)
COMMAND MASK	(Print or set specified mask)
COMMAND MASKNAME	(Parse a mask name from a string)
COMMAND MASKS	(Print names of current masks)
COMMAND MAXLENGTH	(Print length of longest line)
COMMAND MAXNEXT	(Print distance to EOF)
COMMAND MERGE	(Physically combine mutually exclusive lines)
COMMAND MODE	(Set or print mode of file)
COMMAND MOVEFROM	(Move the range after current line)
COMMAND NAME	(Set or print name, type and mode of file)
COMMAND NEXT	(Go to the next line closer to EOF)
COMMAND NORESTORE	(Don't restore saved edit values upon return)
COMMAND OVERLAY	(Change current line with characters specified)
COMMAND PFKEY	(Set or print a PF-key)
COMMAND PFOFF	(Clear PF-key)

COMMAND PFRENEW	(Restore PF-keys)
COMMAND POPMASK	(Change default mask name)
COMMAND PRINT	(Print lines in specified range)
COMMAND PRY	(Enter PRY for debugging)
COMMAND PUSHMASK	(Change default mask name)
COMMAND READ	(Read lines and execute them as edit commands)
COMMAND RECLAIM	(Find all available storage)
COMMAND REMEMBER	(Set or print number of commands to be remembered)
COMMAND REPLACE	(Replace current line)
COMMAND RESOLVE	(Print interpretation of command)
COMMAND RESOLVEORDER	(Set or print order in which commands will be resolved)
COMMAND RESOLVETOKEN	(Print interpretation of command verb)
COMMAND RESTRICTMASK	(Make specified mask more restrictive)
COMMAND RESUME	(Resume editing a file)
COMMAND RETRIEVE	(Print past command)
COMMAND REVERSE	(Execute command in reverse mode)
COMMAND SAVE	(Save current file on disk)
COMMAND SAVED	(Set or print memory of whether file has been saved)
COMMAND SAVEVIEW	(Save current view of file)
COMMAND SAY	(Print message in message area)
COMMAND SCALEX	(Stack a line like the one above input area)
COMMAND SCOPE	(Set or print scope modes or force screen refresh)
COMMAND SCREEN	(Print contents of screen)
COMMAND SCROLL	(Go to a line based on display)
COMMAND SHOUT	(Print warning message in message area and sound alarm)
COMMAND SIMPLIFY	(Print simplification of Boolean expression)
COMMAND SORTX	(Sort lines in specified range)
COMMAND STACK	(Stack lines in specified range)
COMMAND STATUS	(Go to line with specified status or print status)
COMMAND STORE	(Define value of P-EDIT variable)
COMMAND SUPERIMPOSE	(Change current line with characters specified)
COMMAND SYNONYM	(Define a synonym or abbreviation for command)
COMMAND TABKEY	(Set or print PF-key for tab character)
COMMAND TABS	(Set or print tab stops)
COMMAND TAG	(Tag current line with specified name)
COMMAND TAGNAME	(Parse a tag name from a string)
COMMAND TAGS	(Print tags defined in range)
COMMAND TOKEN	(Parse a token from a string)
COMMAND TOP	(Go to the TOP dummy line)
COMMAND TRANSINX	(Print or set input translation tables)
COMMAND TRANSOUTX	(Print or set output translation table)
COMMAND TRUNC	(Set or print length of lines for INSERT)
COMMAND TYPE	(Set or print file type of file)
COMMAND UNDO	(Undo effect of some remembered commands)
COMMAND UNGO	(Go back to current line as of a remembered command)
COMMAND UNMASK	(Clear specified mask)
COMMAND UNSTORE	(Clear P-EDIT variable)
COMMAND UNSYNONYM	(Clear a synonym for a command)
COMMAND UNTAG	(Clear specified tag)
COMMAND UP	(Go to the next line closer to TOP)
COMMAND VARNAME	(Parse a variable name from a string)
COMMAND VARS	(Print names of defined edit variables)
COMMAND VERIFY	(Set verify mode used for typewriters)
COMMAND VERSION	(Print current P-EDIT version)
COMMAND VERSIONS	(Print Boolean expression defining versions of range)
COMMAND VIEW	(Change or print view being displayed)
COMMAND WIDTH	(Set or print width of file on disk)
COMMAND WINDOW	(Set or print part of lines to be displayed)
COMMAND X	(Execute or define command abbreviation)
COMMAND XFERCONTROL	(Use masks from another file)
COMMAND XMACRO	(Execute specified EXEC2 file)
COMMAND Y	(Execute or define command abbreviation)
COMMAND ZONE	(Set or print part of lines to be seen by CHANGE)

CMS COMMANDS

MACRO	$	(Execute CMS command)
MACRO	_	(Execute CMS command and wait for CANCEL (MORE...))
COMMAND	APPEND	(Append current file to existing file on disk)
MACRO	ARCV	(Archive file designated by current line)
MACRO	BURN	(Stop editing current file and erase it on disk)
MACRO	CANCEL	(Leave P-EDIT with no protection for unsaved files)
COMMAND	CMS	(Enter CMS subset operation)
MACRO	COMPARE	(Generate a file showing differences between two files)
MACRO	CONSOLE	(Edit console spool file)
MACRO	CP	(Execute a CP command)
MACRO	CPLIST	(Edit a list of spool files)
MACRO	DISKS	(Edit file showing accessed disks)
MACRO	DROPDISK	(Drop disk designated by current DISKS line)
COMMAND	EDIT	(Read and edit a file)
MACRO	ERASE	(Erase file designated by current line)
MACRO	FILE	(Stop editing current file after saving it on disk)
MACRO	FILELIST	(Generate a list of files on disk)
COMMAND	GETFILE	(Read a file and insert it into current file)
MACRO	GETFILES	(Read a number of files and insert them into current file)
MACRO	GETSYNS	(Insert SYNONYM commands needed by macros)
MACRO	GETTAIL	(Read last lines of a file and insert it into current file)
MACRO	HIST	(Add line to the MAINTAIN HIST file for current file)
MACRO	HUFF	(Encodes file designated by current LIST line)
MACRO	INSTANCE	(Generate an instance of a FORM file)
MACRO	LINKS	(Print users linked to disk)
MACRO	MORE	(Show further detail of COMPARE file)
MACRO	OUT	(Stop editing all saved files)
MACRO	ORDER	(Re-ORDER spool files designated by CP-LIST lines)
MACRO	P-EDIT	(Edit or resume a file or edit a FILE LIST)
MACRO	P-LIST	(Read a group of files for editing)
MACRO	PAUSE	(Save all files on disk that need to be saved)
MACRO	PEDIT	(Edit a file using the PEDIT editor)
MACRO	PUFF	(Decodes file designated by current LIST line))
MACRO	PUTBOOL	(Write Boolean expression on disk)
MACRO	QCOPY	(Copy files designated by LIST file)
MACRO	QMOVE	(Move files designated by LIST file)
MACRO	REACCESS	(Re-access disk designated by current DISKS line)
MACRO	RECEDIT	(Start editing file recursively)
MACRO	RENAME	(Rename file designated by LIST file)
MACRO	RENEW	(Renew current line of LIST file)
MACRO	SAME	(Insert similar file descriptions in LIST file)
COMMAND	SAVE	(Save current file on disk)
MACRO	SAVEFIX	(Save one of the current versions, interactive)
MACRO	SAVEFORM	(Fill in and save one of the current versions, interactive)
COMMAND	SAVEVIEW	(Save current view of file)
MACRO	UNUPDT	(Produce UPDATE file from parametric file)
MACRO	UPDATE	(Apply UPDATE file to current file)
MACRO	WHO	(Print information about who and what is running)
COMMAND	XMACRO	(Execute specified EXEC2 file)

COMMAND MEMORY see MEMORY

CURRENT FILE see CURRENT LINE

CURRENT LINE

The concepts of the current line and current file are central to P-EDIT as it is to most text editors that evolved from typewriter based editors. At any time, one line of current file being edited is designated the current line. Most P-EDIT commands have an implicit argument of the current line. For example, the CHANGE command either modifies the current line alone or a range of lines starting with the current line. Similarly, the LOCATE command searches for the next occurrence of a piece of text following the current line. The current line is always displayed on line 8 on a 3270 display and is often highlighted.

In order to permit the current line to be entirely before or after the lines in the file, a dummy line is placed at each end called the TOP and EOF dummy lines. If the current line is such a dummy line, many commands act in a special fashion, described for each. In any case, the dummy lines are never permitted to be modified or deleted and no line is permitted to be inserted beyond the limits of the file.

related documentation

CONCEPT POSITION COMMANDS	(P-EDIT commands that control current line)
COMMAND HILITE	(Controls method of highliting current line)

CURRENT MASKS see PARAMETRIC FILES

DEFAULT VALUES see INITIALIZATION

DEFAULTS see INITIALIZATION

DICT see DICTIONARY

DICTIONARY

built-in command
    A P-EDIT command, not a macro.

EOF
    The dummy line maintained after all the lines in the file (status is EOF).

excluded line
    A line which is not in any versions of the file that are defined by the current masks (status is TOP, EOF, DUMMY or EXCLUDED).

fixed line
    A line which exists in all versions of the file that are defined by the current masks (status is FIXED).

go to a line
: This refers to the establishment of a different line as the current line of a file.
See: CONCEPT CURRENT LINE

'Not found' is reported
: If the issuer of the command was either the user or a READ command in a macro, 'Not found' is printed (on a 3270 display device, it is displayed in the information area). If the issuer was a macro, a return code of 1 is returned.

printed (stacked) information
: If the issuer of the command was either the user or a READ command in a macro, the information is printed (on a 3270 display device, it is displayed in the information area). If the issuer was a macro, it is put into the CMS program stack (FIFO).

range of lines
: Many P-EDIT commands can operate over a range of consecutive lines. The first line is the current line unless otherwise specified. The extent is specified as a tag which names the last line, an '*' for the remainder of the file, or an unsigned integer defining the number of lines in the range. The first line counts toward the count even if it is excluded, but subsequently only lines in the logical file are counted. An optional range is typically shown in documentation as:

```
<tag>
n
*
```

status of a line
: Either FIXED, TOP, EOF, DUMMY, or for parametric files, UNFIXED, HIDDEN or EXCLUDED.
See: CONCEPT STATUS

TOP
: The dummy line maintained before all the lines in the file (status is TOP).

unfixed line
: A line which is in some, but not all, of the versions of the file defined by the current masks (status is UNFIXED or HIDDEN).

DIRECTION

Most P-EDIT commands for which it makes sense, can be made to operate in the opposite direction in the file by preceding them with '-'. The only exceptions are disks operations such as SAVE since it seems dangerous and not useful to reorder a file as it moves in and out of memory. Typically, the meaning of the opposite direction is that the command operates as though the file were completely reversed, the command were executed, and the file were reversed again (that is its order were restored).

This is the way all primitive P-EDIT commands work. Some macros, however, make an analogous, but not identical, interpretation. In descriptions of command syntax, one will find '<->' preceding the syntactic form of the command when it is influenced by direction. If it follows the normal interpretation of reversal, no further mention is made; it is left to the reader to change 'before' to 'after', 'TOP' to 'EOF', etc. If, however, some different interpretation is made, it will be explained.

P-EDIT macros always operate in non-reversed mode. If they wish to make some interpretation of reversal, they must test for it with the DIRECTION command.

related documentation

COMMAND -	(Prefix to cause reversal)
COMMAND REVERSE	(Same as '-' for macros)
COMMAND DIRECTION	(Testing reversal for macros)

DISK see MAINTENANCE

DISPLAY COMMANDS

COMMAND	?	(Print the last few commands)
MACRO	BOOL	(Print Boolean expressions for lines in range)
COMMAND	CALLER	(Test who caller was)
COMMAND	COLUMN	(Print location of string in current line)
COMMAND	DESERIALIZE	(Print deserialize mode and columns)
COMMAND	DIRECTION	(Test for reverse mode by caller)
COMMAND	EDIT	(Print list of files being edited)
COMMAND	EDITING	(Print file number of file specified)
COMMAND	EDITMODE	(Print whether to edit text or Boolean expressions)
COMMAND	FCOUNT	(Print number of files being edited)
COMMAND	FETCH	(Print value of edit variable)
COMMAND	FNUMBER	(Print number of file and controlling file)
COMMAND	FORMAT	(Print format of current file)
COMMAND	FREE	(Print amount of available storage)
COMMAND	HEAD	(Set header line of screen)
COMMAND	HEADVARS	(Set or print order of resolution for header line)
MACRO	HELP	(Same as TELL TELL)
MACRO	HIDE	(Change view to hide current (unfixed) line)
COMMAND	IMAGE	(Print handling of tab and back-space characters)
COMMAND	LENGTH	(Print length of current line)
COMMAND	LEVEL	(Print level of editing)
COMMAND	LINES	(Print physical line number and number of lines)
COMMAND	MACROLINE	(Stack calling command)
COMMAND	MASK	(Print value of specified mask)
COMMAND	MASKS	(Print names of current masks)
COMMAND	MAXLENGTH	(Print length of longest line)
COMMAND	MAXNEXT	(Print distance to EOF)
COMMAND	MODE	(Print mode of file)
COMMAND	NAME	(Print name, type and mode of file)
MACRO	NOTE	(Find previous note)
MACRO	P-EDIT	(Edits a file of files being edited)
MACRO	P-LIST	(Edits a file of files being edited or a subset of that)
MACRO	P-SCROLL	(Special scroll for TELL and LISP files)
RESOLVE	PAREN	(Check for balanced parentheses)
COMMAND	PFKEY	(Print value of specified PF-key)
COMMAND	PRINT	(Print lines in specified range)
COMMAND	RESOLVE	(Print interpretation of command)
COMMAND	RESOLVETOKEN	(Print interpretation of command verb)
COMMAND	RESOLVEORDER	(Print order in which commands will be resolved)
MACRO	RETCODE	(Print the return code that would have been returned)
COMMAND	RETRIEVE	(Print past command)
MACRO	REVEAL	(Display the lines the satisfy specified condition)
MACRO	RIGHT	(Move WINDOW right)
COMMAND	SAVED	(Print memory of whether file has been saved)
COMMAND	SAY	(Print message in message area)
COMMAND	SCALEX	(Stack a line like the one above input area)
COMMAND	SCOPE	(Print scope modes or force screen refresh)
COMMAND	SCREEN	(Print contents of screen)
COMMAND	SCROLL	(Go to line based on display)
COMMAND	SHOUT	(Print warning message in message area and sound alarm)
MACRO	SHOW	(Start loop to show all possible versions of range)
MACRO	SHOWMASK	(Edit a file showing current masks)
MACRO	SHOWPF	(Edit a file showing current PF-key settings)
MACRO	SHOWSYNS	(Edit a file showing current P-EDIT synonyms)
MACRO	SHOWPARM	(Edit a file parameters and values controling lines)
MACRO	SHOWVARS	(Edit a file showing current edit variables)
MACRO	SHOWVERS	(Edit a file showing current versions of file)
MACRO	SPELL	(Detect incorrectly spelled lines and offers corrections)
COMMAND	STACK	(Stack lines in specified range)
COMMAND	STATUS	(Print or go to line with status relative to current masks)
COMMAND	SYNONYM	(Define a synonym or abbreviation for command)
COMMAND	TABKEY	(Print PF-key for tab character)
COMMAND	TABS	(Print tab stops)
COMMAND	TAGS	(Print tags defined in range)
MACRO	TELL	(Edit a file to explain specified part of P-EDIT)
COMMAND	TRANSINX	(Print input translation tables)
COMMAND	TRANSOUTX	(Print output translation table)
COMMAND	TRUNC	(Print length of lines for INSERT)
COMMAND	TYPE	(Print type of file)
MACRO	UNHIDE	(Change current view to exclude current line)

COMMAND	VARS	(Print names of defined edit variables)
COMMAND	VERIFY	(Print verify mode used for typewriters)
COMMAND	VERSION	(Print current P-EDIT version)
COMMAND	VERSIONS	(Print Boolean expression defining versions of range)
COMMAND	VIEW	(Print or change view being displayed)
MACRO	WHO	(Print information about who and what is running)
COMMAND	WIDTH	(Print width of file on disk)
COMMAND	WINDOW	(Print columns of lines to be displayed)
COMMAND	ZONE	(Print part of lines to be seen by CHANGE)

DISPLAY DEVICE see SCREEN FORMAT

EDIT VARIABLES see VARIABLES

EDITOR DISK see MAINTENANCE

FILE COMMANDS

MACRO	BLIND	(Enter INPUT mode with SCOPE OFF)
COMMAND	CASE	(Set upper-case folding mode)
COMMAND	DESERIALIZE	(Set deserialize mode and columns)
MACRO	FIELDBR	(Set current fill-in field brackets)
COMMAND	FORMAT	(Set format of file (V, F, etc.))
COMMAND	HILITE	(Set method of highlighting current and unfixed lines)
COMMAND	IMAGE	(Set handling of tab and back-space characters)
MACRO	INDENT	(INPUT that automatically lines up line with previous)
COMMAND	INPUT	(Enter batch INPUT mode)
COMMAND	MODE	(Set mode of file)
COMMAND	NAME	(Set name, type and mode of file)
MACRO	NOTE	(Set a note at current line or find previous note)
MACRO	PAUSE	(Save all files on disk that need to be)
COMMAND	PFKEY	(Set a PF-key)
COMMAND	PFOFF	(Clear PF-key)
COMMAND	PFRENEW	(Restore PF-keys)
COMMAND	SAVE	(Save current file on disk)
COMMAND	SAVED	(Set memory of whether file has been saved)
MACRO	SETUP	(Set file values as would be done for specified file type)
COMMAND	TABKEY	(Set PF-key for tab character)
COMMAND	TABS	(Set tab stops)
COMMAND	TRUNC	(Set length of lines for INSERT)
COMMAND	TYPE	(Set file type of file)
COMMAND	VERIFY	(Set verify mode used for typewriters)
COMMAND	WIDTH	(Set width of file on disk)
COMMAND	WINDOW	(Set part of lines to be displayed)
COMMAND	ZONE	(Set part of lines to be seen by CHANGE)

GARBAGE COLLECTION see RECLAMATION

GARBAGE COLLECTOR see RECLAMATION

GLOBAL COMMANDS

COMMAND	ALARM	(Print or set permitting beeping on errors)
COMMAND	BROWSE	(Print or set BROWSE mode which disallows changes)
COMMAND	EXIT	(Stop execution of READ *)
RESOLVE	FIRST	(Set default FIRST tag for MOVE, etc.)
MACRO	FORGET	(Clear remembered commands)
COMMAND	GO	(Go to line named by the specified tag)
COMMAND	HEADVARS	(Set order of resolution for header line)
RESOLVE	LAST	(Set default LAST tag for MOVE, etc.)
MACRO	RANGE	(Set default FIRST or LAST tag for MOVE, etc.)
COMMAND	RESOLVEORDER	(Print or set order in which commands will be resolved)
COMMAND	RESUME	(Resume editing a file)
MACRO	SHORT	(Define abbreviation for command)
MACRO	SIMEDIT	(Permit input and commands to be intermixed)
COMMAND	STORE	(Define value of P-EDIT variable)
COMMAND	SYNONYM	(Define a synonym or abbreviation for command)
COMMAND	TAG	(Tag current line with specified name)
RESOLVE	TO	(Set default TO tag for MOVE, etc.)
MACRO	TRANSIN	(Set input translation)
COMMAND	TRANSINX	(Print or set input translation tables)
MACRO	TRANSOUT	(Set output translation)
COMMAND	TRANSOUTX	(Print or set output translation table)
COMMAND	UNSTORE	(Clear P-EDIT variable)
COMMAND	UNSYNONYM	(Clear a synonym for a command)
COMMAND	UNTAG	(Clear specified tag)
COMMAND	X	(Execute or define command abbreviation)
COMMAND	Y	(Execute or define command abbreviation)

HC see INTERRUPTS

IMAGE see IMAGING

IMAGING

Different types of files require different processing of text as it is entered into the file. This is called imaging; there are three options: ON, CANON and OFF.

ON: Text which is given in the form of lines (FIND, INSERT, OVERLAY and REPLACE) is "expanded" into a "line image" in which back-spaces are removed and tabs are replaced by the appropriate number of blanks. The procedure which builds this line image simulates a typewriter having columns 1 to the current TRUNC column the appropriate tab stops. If the input line contains back-spaces, the column pointer may move forwards and backwards.

Characters in the line image are not affected by being backspaced over.

Characters which are moved over forwards are replaced.

Truncation is deemed to occur if a non-blank character is left in the line beyond the column of truncation for the file. (In the case of OVERLAY, '_' is not considered a non-blank.)

Text which is not given as an entire line (CHANGE, LOCATE and ALTER) is not expanded, that is, tabs and back-spaces are treated in the same way as other characters.

CANON: Backspaces may be used to produce compound characters, as in underlining. Before they are inserted in the file, these compound characters are put into a "canonical" form which is independent of the order in which their components were given. (Backspaces are arranged singly between the characters which overlay each other, and the overlaying characters are arranged according to their EBCDIC values.) Tab characters do not receive special treatment, and enter the file in the same way as ordinary printing characters. Any attempt to canonicalize more than 255 characters will fail.

OFF: Tabs and back-spaces are treated in the same way as other characters, that is, they enter the file without being "expanded" or reordered.

See: COMMAND IMAGE

INITIALIZATION

When P-EDIT is first entered and every time the editing of a file is started, the user is given the opportunity to execute an EXEC2 files to set various defaults. Although their file type is not 'P-MACRO', these EXEC2 files are executed in the same fashion as P-EDIT macros.

When P-EDIT is first entered, it executes the file SETUP P-EXEC. This file can define synonyms, character translation tables, PF-keys and the like. There is a standard SETUP P-EXEC provided upon which many standard macros depend. A file named PROFILE P-EXEC can be used to provide additional setup. Files named TERMINAL P-EXEC and PFKEYS P-EXEC can be used to provide alternative setup of the terminal and PF-keys. It is recommended that users use this method to personalize P-EDIT, rather than modify the standard SETUP file.

See: EXEC SETUP	(The standard SETUP P-EXEC)
MACRO GETSYNS	(Inserts the standard SYNONYM definitions)

The following commands will, if issued during initialization, establish default values for the entire edit session in addition to their normal function. All but the first two are values that are associated with a file. The first two are associated with the editing level.

COMMAND REMEMBER	(Number of commands to remember)
COMMAND RESOLVORDER	(Resolution order for user issued commands)
COMMAND HILITE	(Highlighting of current line and UNFIXED lines)
COMMAND PFKEY	(Setting of PF-keys)
COMMAND SCOPE	(Display of long lines and undisplayable characters)
COMMAND TRANSINX	(Translation of characters when read)
COMMAND TRANSOUX	(Translation of characters when displayed)
COMMAND VERIFY	(Printing changed and located lines)

description of file initialization

Whenever the editing of a file starts (unless the NOSETUP option is specified), P-EDIT tries to find the following files:

```
P-SETUP  P-EXEC  A
filetype P-SETUP *
DEFAULT  P-SETUP *
```

The first one found is executed prior to reading the file to be edited. This file can set any file values it wishes by issuing the appropriate P-EDIT commands. If the command desired cannot be issued until the edited file is read in (such as MASK), the P-SETUP file can stack it; any such stacked lines are executed immediately after reading in the edited file.

The standard DEFAULT P-SETUP first looks to see if the first four characters of the file are 'UPDT' and a file UPDT P-SETUP exists; if so it executes it. If not, it tries to look up the edited file type in the file SETUP P-TABLE; if found, it uses the information there for initialization. If not, it uses default values built into it.

INTERRUPT see INTERRUPTS

INTERRUPTS.

There are two kinds of interrupts in P-EDIT, out-of-storage and user. An out-of-storage interrupt occurs when P-EDIT no longer has enough storage to operate (or when a macro returns with a return code of -9). A user interrupt occurs when a line containing only the characters 'HC', for Halt Command, is entered (or a macro returns with a return code of -10). When either type of interrupt occurs, the currently running command is terminated and its effects are undone.

If the command was issued by the user, one of the following warnings is printed:

```
No storage -- suppressed <terminated command>
User interrupt -- suppressed <terminated command>
```

If a macro was executing and it can not be undone because it issued a REMEMBER command, one of the following alternate warnings is printed:

```
No storage -- aborted <macro call>
User interrupt -- aborted <macro call>
```

Some macros that modify the CMS file system do this on purpose so that the user is warned of the irrevocable modifications that have been done.

treatment of interrupted subcommand

If the interrupted command was issued by a macro (a subcommand) the appropriate return code is returned, -9 or -10. After undoing the partial effects of the subcommand, one of these warning messages is generated:

```
No storage ignored by <EXEC2 file name>
User interrupt ignored by <EXEC2 file name>
```

However, if the macro responds to the return code as expected, by &EXITing with the same return code, the warning will be undone as a natural consequence of the undoing of the effects of the macro.

See:	CONCEPT MACROS	(P-EDIT macros)
	MACRO SAMPLE	(Sample macro)
	CONCEPT RECLAMATION	(Treatment of storage)

JARGON see DICTIONARY

LOGICAL FILE see PARAMETRIC FILES

MACRO see MACROS

MACROES see MACROS

MACROS

The facilities that are built-in to P-EDIT are very primitive, bordering on unusable -- it is macros that combine these primitive commands into the environment users actually use. Users are encouraged to write their own macros and to make them generally available to the community. Macros are written in the EXEC2 language, the same language many use to extend the CMS command environment. P-EDIT macros are expected to conform to certain conventions described shortly. Familiarity with EXEC2 will be assumed. Information about EXEC2 is available in Yorktown report RC7268 (on-line as EXEC2 MEMO on CMS19F) and on a reference summary card.

how to define macros

P-EDIT macros are CMS files with a file type of 'P-MACRO'. The file name is the macro name that the user would use like a P-EDIT command name. When used, the remainder of the line is divided into words, separated by blanks, and each word is assigned to the EXEC2 variables &1, &2, etc. (&0 is assigned to the command name actually used -- this might be different than the file name if a synonym, defined with the SYNONYM command, was used.) If the macro needs a different parsing of its arguments, the uninterpreted argument string is available as &ARGSTRING; a number of P-EDIT commands are provided for parsing this in various common ways. For flexibility, arguments are passed in the case they were entered; many macros use the &UPPER statement to convert them to upper-case.

See: CONCEPT INITIALIZATION	(How to automatically have synonyms defined)
CONCEPT MAINTENANCE	(How to make macros generally available)
COMMAND SYNONYM	(Defining synonyms)
COMMAND TOKEN	(Parsing edit tokens and delimited strings)
COMMAND TAGNAME	(Parsing tag, variable and mask names)

description of resolution order

EXEC2 will presume that any line that is not an EXEC2 statement is a P-EDIT command and will pass it on to P-EDIT. Thus the &COMMAND statement must be used to issue CMS commands. When P-EDIT receives the command from EXEC2, it does not check its list of synonyms, defined with the SYNONYM command, as it normally does first. Rather, it immediately looks to see if it is a built-in P-EDIT command; if not, it tries to find a file of type P-MACRO whose name matches the command name used. Built-in abbreviations (such as N for NEXT) are respected during this process. A number of commands are available for modifying this resolution order (to restore the normal order, the macro could begin with: RESOLVORDER SYNONYM COMMAND MACRO). The EXEC2 statement &PRESUME can be used to change the presumption that commands are to be sent to P-EDIT.

See: CONCEPT RESOLUTION

description information restored on exit

Macros commonly change various file characteristics in their operation. It is usually the intention that these changes are temporary. P-EDIT automatically saves and restores the values unless the macro issues the command NORESTORE or a different file has become the current file. The following commands set the values that are saved and restored:

COMMAND CASE
COMMAND DESERIALIZE
COMMAND EDITMODE
COMMAND IMAGE
COMMAND TABS
COMMAND TRUNC
COMMAND VERIFY
COMMAND WIDTH
COMMAND WINDOW
COMMAND ZONE

how macros receive information normally printed

A couple of techniques operate to solve the problem that information cannot be communicated to a macro in the same way it could to the user, by printing it on the display device. Instead of printing error messages, non-zero return codes are returned (EXEC2 variable &RC):

-10	User interrupt, 'HC', occurred during command
-9	P-EDIT ran out of storage during command
-1	Invalid P-EDIT command
1	'Not found' or at EOF (TOP) dummy line
2	Error during execution of the command
4	Text truncated or line greater than 254 characters stacked
5 to 7	Used for SIMEDIT, see CONCEPT RESOLUTION
8	CMS command error, by convention

Commands that are used to print information, stack their result instead of printing it (ZONE). Commands that normally print an indefinite number of lines, stack a null line after stacking the informational lines (VARS).

description of handling of stack

The CMS stack is the major vehicle of communication between macros and P-EDIT. Before starting a macro execution, P-EDIT makes a new buffer in the stack so the macro need not worry about irrelevant lines in the stack. When the macro returns, P-EDIT drops this buffer and any subsequent ones that were made. If the purpose of a macro is to leave a line in the stack, it should issue the CMS command DROPBUF before stacking that line.

Lines in the CMS stack are restricted to 255 characters. Because files can be as wide as 65535 and because Boolean expressions are often quite long, it is necessary to establish a convention for handling long lines. When P-EDIT tries to stack a line longer than 254, it breaks it up into 255 long segments terminated with a line shorter than 255 unless the string is the maximum length (this might result in a null line to terminate segments). When this happens, the command will return with a return code of 4. Conversely, if P-EDIT reads a line which is 255 in length, it assumes it is a segment of a long line and concatenates consecutive lines until it finds one smaller than 255.

required documentation

Each macro snould be documented with a TELL. TELLs are defined by having a line within the first 5 lines of the macro with the line (*TELL LABEL -label). The text of the TELL is between the specified label and the next label. Screens must be separated by lines beginning with '-' and be no longer than 17 lines (18 for first and last, 19 if only one screen). These lines may contain a comment if desired. The text of the TELL should be no longer than 79 characters wide.

how to handle interrupts

P-EDIT has two types of interrupts, out-of-storage and user initiated (the named interrupt, 'HC'). One of these two could occur during the execution of a macro. In the best of all possible worlds, P-EDIT would grab control from EXEC2, undo the partial effects of the macro, and report the interrupt to the user -- just like it does for a built-in command. But EXEC2 doesn't permit this, so macros must detect interrupts and exit back to P-EDIT.

Whenever a command issued by a macro runs out of storage, it returns with a return code of -10; whenever a user interrupt is detected, it returns with a -9. If the macro exits with of these return codes, P-EDIT will undo the effects of the macro and tell the user. The easiest way for a macro to fulfill this responsibility is to issue the statement: &ERROR &IF &RC <= -9 &EXIT &RC. Care must be taken when issuing CMS commands that their return code is not misunderstood should less than or equal to -9. These return codes were chosen not to conflict with by CMS commands typically used by P-EDIT macros.

See: MACRO SAMPLE (A good base for writing macros)

how to call P-EDIT recursively

There are a set of conventions needed to make the file ring structure of P-EDIT match the stack structure of EXEC2 invocations. If a macro wishes to permit the user to enter commands but not give up control, it should use the macro RECEDIT (RECursive EDIT) or embed the equivilant statements and use the READ command. Blind use of READ * will confuse macros such as QUIT and FILE and ultimately the user.

See:	MACRO RECEDIT	(Used to call P-EDIT recursively)
	COMMAND READ	(Used to restart reading commands from user)

relation of parametric file and macros

In parametric files, a user is warned that he is about to edit unfixed lines by having them highlighted. A macro, not having eyes, must use commands to test this explicitly. Macros basically have three choices:

1) They can refuse to operate on unfixed lines. In this case they should use the STATUS command to check conformance.

2) They can operate as though some line is fixed. In this case they should use the FIX command to set a mask and PUSHMASK and POPMASK to keep from destroying the user's masks.

3) They can go into a loop, doing their thing once for every possible instantiation of a range of lines. In this case they should use the INSTANCES command to stack the Boolean expression for each instantiation and use PUSHMASK, MASK and POPMASK to establish the necessary masks.

See: CONCEPT PARAMETRIC FILES

MAINTAIN see MAINTENANCE

MAINTENANCE

P-EDIT is kept on the disk that is labeled EDITOR along with a number of other text editors. EDITOR is AUTOPROG 192 on both the V and T-machines. It is maintained by Walt Daniels using the MAINTAIN EXEC he developed. Users wishing to add P-EDIT macros or setup files to this disk should use MAINTAIN to do so. Further details can be obtained by issuing the command: _MAINTAIN ?.

See:	CONCEPT MACROS	(How to define macros)
	CONCEPT INITIALIZATION	(How to define setup files)

MASK see PARAMETRIC FILES

MASKS see PARAMETRIC FILES

MEMORY

P-EDIT has the ability of remembering any number of prior commands and what effects they had. A number of facilities use this memory to permit the user to see previous commands, re-execute them, remove the effects of them, and the like. Of course, the effects of commands that are out of P-EDIT's control, such as modifications to CMS files, or which have already occurred, such as printing messages, cannot be undone.

The COVERT prefix can be used to prevent the remembering of a command (although its effects are still remembered as additional effects of the previous command). The following commands are automatically executed covertly: ?, " and the null command.

levels of memory

Commands are remembered in levels. A new level is started whenever the execution of a macro begins or a new level of editing starts (READ *). When the level ends (&EXIT terminates a macro or EXIT terminates an editing level), all the commands that were remembered during that level are wrapped up into one unit of memory and associated with the command that started the level (the macro call or READ command). Thus from the point of view of the user, a macro appears to be a single command, while from the point of view of the macro, each command it executes is a separate command.

Each level of memory has associated with it a limit of how many commands should be remembered (to protect against running out of storage). A level started by a macro has no limit unless the macro explicitly states one, while a level started with a READ command, has the default limit (initially 9, but can be modified by SETUP P-EXEC).

related documentation

CONCEPT	INITIALIZATION	(How to set default limit of memory)
COMMAND	REMEMBER	(Resets memory limit)
MACRO	FORGET	(Clears selected memory)
COMMAND	?	(Print remembered commands)
COMMAND	"	(Re-execute last remembered command)
MACRO	REPLAY	(Re-execute remembered commands)
COMMAND	UNDO	(Undo remembered comm[illegible]ds)
MACRO	BACK	(Undo more remembered commands)
COMMAND	UNGO	(Undo positioning of remembered commands)
CONCEPT	MACROS	(P-EDIT macro facility)
COMMAND	READ	(Create a new editing level)
COMMAND	COVERT	(Absorb specified command into previous)
COMMAND	RETRIEVE	(Used by macros to find remembered commands)

MODIFICATION COMMANDS

MACRO	ADD	(Add string to end of line)
MACRO	ADJUST	(Indent line according to previous line)
COMMAND	ALTER	(Replace one character with another over range)
MACRO	ANNEX	(Insert a line as extension of previous one)
MACRO	ASMCOM	(Adjust comment on line of ASSEMBLE file)
MACRO	BALANCE	(Check for balanced parentheses)
MACRO	BLIND	(Enter INPUT mode with SCOPE OFF)
COMMAND	CHANGE	(Replace one string with another over range)
MACRO	CHINDENT	(Change characters used for indentation)
RESOLVE	CMC	(Mixed-case CHANGE)
RESOLVE	COPY	(Copy lines from one point to another)
MACRO	CUF	(Replace unique occurence of string in file as specified)
MACRO	CUS	(Replace unique occurence of string on screen)
COMMAND	COPYFROM	(Copy the range to after the current line)
MACRO	CUT	(Cut current line into two lines)
COMMAND	DELETE	(Delete lines in range)
MACRO	DOTHERS	(Delete unselected lines)
MACRO	DTOP	(Delete lines from previous to the top)

MACRO	DUP	(Duplicate lines)
MACRO	EMBED	(Surround current line with specified text)
MACRO	EXCHANGE	(Exchange specified strings)
MACRO	FILLIN	(Fill in fill-in field with string)
COMMAND	GETFILE	(Read a file and insert it into current file)
MACRO	GETFILES	(Read a number of files and insert them into current file)
MACRO	GETTAIL	(Read last lines of a file and insert it into current file)
MACRO	GETSYNS	(Insert SYNONYM commands needed by macros)
MACRO	HEX	(Insert copy of current line translated to hexidecimal)
RESOLVE	HYPHENATE	(Move piece of last word to next line)
MACRO	INCHANGE	(Interactive CHANGE)
MACRO	INDENT	(INPUT that automatically lines up line with previous)
MACRO	INFILLIN	(Interactive FILLIN)
COMMAND	INPUT	(Enter batch INPUT mode)
COMMAND	INSERT	(Insert a new line after current line)
RESOLVE	INSPELL	(Interactively corrects spelling of file)
MACRO	INVERT	(Invert order of lines)
MACRO	K	(Split a line into two lines)
RESOLVE	KC	(Combine two lines into one)
MACRO	LCASE	(Force text to be in lower case)
MACRO	LINKS	(Insert lines showing users linked to disk)
COMMAND	MAKE	(Modify Boolean expressions over the range)
MACRO	MAKESAME	(Make parametric file out of two non-parametric ones)
COMMAND	MERGE	(Physically combine mutually exclusive lines)
MACRO	MOVE	(Move lines from one point to another)
COMMAND	MOVEFROM	(Move the range after current line)
RESOLVE	MOVEWORDS	(Move last words to beginning of next line)
COMMAND	OVERLAY	(Change current line with characters specified)
MACRO	PARA	(Format a paragraph of text)
MACRO	PASTE	(Combine two lines, opposite of CUT)
MACRO	PROMOTE	(Force current line to exist in all current versions)
MACRO	PURGE	(Remove unwanted versions of parametric file)
MACRO	REINPUT	(Execute command and re-enter INPUT mode)
MACRO	RENEW	(Renew current line of LIST file)
COMMAND	REPLACE	(Replace current line)
MACRO	RESERIAL	(Reserialize lines in range)
MACRO	RETROFIT	(Combine two similar parametric files)
MACRO	SAME	(Insert similar file descriptions in LIST file)
MACRO	SAVEFORM	(Fill in and save one of the current versions, interactive)
MACRO	SCALE	(Insert a scale line like the one on top of input area)
MACRO	SETCOL	(Change specified columns to specified string)
MACRO	SORT	(Sort lines in specified range)
COMMAND	SORTX	(Sort lines in specified range)
MACRO	SUBST	(CHANGE with only beginning and end of text specified)
COMMAND	SUPERIMPOSE	(Change current line with characters specified)
MACRO	TRIM	(Shorten the lines over the range if too long)
RESOLVE	UCASE	(Force text to be in upper case)
MACRO	UNSERIAL	(Blank serialization field over range)
MACRO	UPDATE	(Apply UPDATE file to current file)

NAMED INTERRUPT see INTERRUPTS

NO STORAGE see INTERRUPTS

OTHER COMMANDS

COMMAND	*	(Do nothing)
COMMAND	"	(Execute previous command again)
MACRO	AGAIN	(Execute remembered command again)
MACRO	ARGS	(Execute command after setting arguments in current line)
MACRO	BACK	(Undo more (or less) than the previous UNDO or BACK)
COMMAND	COMMAND	(Execute built-in command)
COMMAND	COVERT	(Execute command as part of previous one)
COMMAND	DELIMSTRING	(Parse a delimited string from a string)
MACRO	DUMMY	(Create a dummy line)
COMMAND	ERROR	(Act like an illegal command)
MACRO	EVAL	(Evaluate arithmetic expression)
MACRO	FOR	(Execute command for each line of range)
MACRO	HEX	(Insert copy of current line translated to hexidecimal)
MACRO	IE	(Execute commands accidentally inserted)
MACRO	IF	(Conditional execution of edit commands)
COMMAND	MACRO	(Execute P-EDIT macro)
COMMAND	MASKNAME	(Parse a mask name from a string)
MACRO	MCASE	(Execute command in mixed-case mode)
COMMAND	NORESTORE	(Don't restore saved edit values upon return)
MACRO	PRESUME	(Enter mode where input is prefixed as specified)
COMMAND	PRY	(Enter PRY for debugging)
COMMAND	READ	(Read lines and execute them as edit commands)
COMMAND	RECLAIM	(Find all available storage)
MACRO	REINPUT	(Execute command and re-enter INPUT mode)
COMMAND	REMEMBER	(Print or set number of commands to be remembered)
MACRO	REPLAY	(Selectively undo remembered commands)
COMMAND	REVERSE	(Execute command in reverse mode)
COMMAND	SIMPLIFY	(Print simplification of Boolean expression)
MACRO	SUM	(Add column of numbers over range)
COMMAND	TAGNAME	(Parse a tag name from a string)
COMMAND	TOKEN	(Parse a token from a string)
COMMAND	UNDO	(Undo effect of some remembered commands)
COMMAND	VARNAME	(Parse a variable name from a string)

OUT OF STORAGE see INTERRUPTS

P-EDIT JARGON see DICTIONARY

P-EDIT VARIABLES see VARIABLES

PARAMETRIC see PARAMETRIC FILES

PARAMETRIC COMMANDS

MACRO	ANNEX	(Insert a line as extension of previous one)
COMMAND	ATBASE	(Set base application as defined by current masks)
MACRO	BOOL	(Print Boolean expressions for lines in range)
MACRO	CONFLICT	(Find next pair of consecutive UNFIXED and HIDDEN lines)
COMMAND	EDITMODE	(Print or set whether to edit text or Boolean expressions)
COMMAND	EXCLUDE	(Restrict mask to exclude range)
COMMAND	EXCLUDEVIEW	(Restrict mask to exclude viewed lines in range)
COMMAND	FIX	(Restrict mask to fix range)
COMMAND	FIXVIEW	(Restrict mask to fix viewed lines in range)
COMMAND	FORMAT	(Make current file parametric)
RESOLVE	GOSHOW	(Go to line at beginning of SHOW loops range)
MACRO	HIDE	(Change view to hide current (unfixed) line)
COMMAND	HILITE	(Set method of highlighting current and unfixed lines)
MACRO	INFIX	(Interactively sets a mask to fix the file)
MACRO	INSTANCE	(Generate an instance of a FORM file)
COMMAND	MAKE	(Modify Boolean expressions over the range)
MACRO	MAKESAME	(Make a parametric file out of two non-parametric ones)
COMMAND	MASK	(Print or set specified mask)
COMMAND	MERGE	(Physically combine mutually exclusive lines)
COMMAND	POPMASK	(Change default mask name)
MACRO	PROMOTE	(Force current line to exist in all current versions)
MACRO	PURGE	(Remove unwanted versions of parametric file)
COMMAND	PUSHMASK	(Change default mask name)
MACRO	PUSHPOP	(PUSHMASK and POPMASK for PF-keys)
MACRO	PUTBOOL	(Write Boolean expression on disk)
COMMAND	RESTRICTMASK	(Make specified mask more restrictive)
MACRO	RETROFIT	(Combine two similar parametric files)
MACRO	SAVEFIX	(Save one of the current versions, interactive)
MACRO	SAVEFORM	(Fill in and save one of the current versions, interactive)
COMMAND	SAVEVIEW	(Save current view of file)
MACRO	SHOW	(Start loop to show all possible versions of range)
MACRO	SHOWMASK	(Edit a file showing current masks)
MACRO	SHOWPARM	(Edit a file parameters and values controling lines)
MACRO	SHOWVERS	(Edit a file showing current versions of file)
RESOLVE	STEP	(Step to next version of SHOW loop)
MACRO	TIME	(Re-set TIME mask)
MACRO	UNFIXED	(Find next occurrence of unfixed line)
MACRO	UNHIDE	(Change current view to exclude current line)
COMMAND	UNMASK	(Clear specified mask)
MACRO	UNSHOW	(Terminate SHOW loop)
COMMAND	VIEW	(Print or change view being displayed)
COMMAND	XFERCONTROL	(Use masks from another file)

PARAMETRIC FILE see PARAMETRIC FILES

PARAMETRIC FILES

P-EDIT's most novel feature permits the viewing and modification of parametric files in a far more convenient fashion than possible previously. A parametric file is a file which contains the necessary information to generate any of a large number of similar files or versions. Each line of text which is common to multiple versions exists only once, thus any change common to multiple versions can be made only once. Each line of the file consists of its text and control information (Boolean expression) which defines which of the versions has that text.

With P-EDIT, the user can select which of the possible versions he wishes to edit; he can even select an entire set of versions to edit simultaneously. Even so, he always views on the screen simply one of the possible versions. All the information he needs to know about other versions he is editing simultaneously, is conveyed by highlighting various lines of the view presented.

definition of real, logical, and viewed files

Real file: A file as it resides on disk is called the real file. The real file contains all the possible versions of that file. Each record consists of text and a Boolean expression, example:

```
                CALL RDBUF(PLIST);                        SYSTEM=VM
```

The Boolean expression defines which versions of the file contain the corresponding text. When a file is edited, the entire real file is read into memory and remains there while it is being edited. The only built-in P-EDIT commands that operate on the real file are STATUS, LINES, and various I/O commands (SAVE and such).

Logical file: The versions of the file which the user wishes to edit at the moment is called the logical file. It is a subset of the real file. This subset is defined by masks which mask out the undesired lines of the real file. Masks are Boolean expressions which have names by which they can be referred. Only the lines in the real file whose Boolean expressions are consistent with the masks are in the logical file. A Boolean expression is inconsistent with the masks if it cannot be true when all the masks are true; such lines are called excluded. The lines that remain in the logical file are called fixed if they occur in all versions of the file still consistent with the masks and are call unfixed if they are in some, but not all, of those versions. Most P-EDIT commands operate on the logical file.

Viewed file: Only one version of the file can be displayed; that version is called the viewed file. The viewed file is one of the versions in the logical file. Which version is an arbitrary choice made by P-EDIT on an as-needed basis unless the user specifies some preference. A line that is not in the viewed file is called hidden. An unfixed line in the viewed file is displayed highlighted to warn the user that some of the versions in the logical file do not contain that line. If a hidden line is surrounded by fixed lines, the previous fixed line is highlighted so that the existence of the hidden line will be known. The nature of the highlighting is determined by the HILITE command. The only built-in P-EDIT command that operates on the viewed file is SCROLL.

See:	CONCEPT BOOLEAN EXPRESSIONS	(Syntax of Boolean expressions)
	CONCEPT MASK NAME	(Syntax of mask names)
	CONCEPT STATUS	(Status of lines relative to current masks)
	CONCEPT PARAMETRIC COMMANDS	(Commands relating to parametric files)
	COMMAND SAVEVIEW	(Saving the current view)

POSITION COMMANDS

MACRO	AVAND	(Go to next line missing any specified string)
COMMAND	AVOID	(Go to next line missing specified string)
MACRO	AVOR	(Go to next line missing all specified strings)
COMMAND	BOTTOM	(Go to the last line in the file)
MACRO	CONFLICT	(Find next pair of consecutive UNFIXED and HIDDEN lines)
COMMAND	FIND	(Go to next line with matches pattern)
MACRO	FINDTAG	(Find next occurrence of tag)
COMMAND	GO	(Go to the line named by the specified tag)
RESOLVE	GOSHOW	(Go to the line ae beginning of SHOW loops range)
MACRO	LAND	(Go to next line with all specified strings)
MACRO	LHEX	(Go to next line with specified characters in EBCDIC)
RESOLVE	LMC	(Go to next line with specified mixed-case string)
COMMAND	LOCATE	(Go to next line with specified string)
MACRO	LONGEST	(Go to the longest line in the file)
MACRO	LOR	(Go to next line with any specified strings)
MACRO	LUF	(Go to line with specified string uniquely in file)
MACRO	LUS	(Go to line with specified string uniquely on screen)
COMMAND	NEXT	(Go to the next line closer to EOF)
MACRO	SEARCH	(Execute command over each of several files)
COMMAND	STATUS	(Go to line with specified status)
COMMAND	TOP	(Go to TOP dummy line)
MACRO	UNFIXED	(Find next occurrence of unfixed line)
COMMAND	UNGO	(Go back to current line as of a remembered command)
COMMAND	UP	(Go to next line closer to TOP)

REAL FILE see PARAMETRIC FILES

RECLAIM see RECLAMATION

RECLAMATION

P-EDIT needs a minimum of 285K to run; this probably means a minimum of a 1.5M virtual machine in practice. The storage is managed with a garbage collection strategy: storage is used until none is left; then a rather expensive operation, reclamation, is used to free available storage. You can avoid an unexpected pause in editing by using the command, RECLAIM, periodically when you realize that you are not going to need to edit for a while.

See: COMMAND	RECLAIM	(Increases available storage)
COMMAND	FREE	(Prints avialable storage)
MACRO	FORGET	(Frees storage used to remember the effect of commands)

execution of RECLAIM P-EXEC

Prior to starting storage reclamation, the file RECLAIM P-EXEC * is executed with &0 set to START and &1 set to ON or OFF depending on the current setting of SCOPE. Prior to executing the next P-EDIT command after reclamation is completed, that same file is executed with &0 set to DELAYED and &1 the same.

The standard RECLAIM P-EXEC updates line 0 of the screen so that the user will know why P-EDIT is not responding (if SCOPE is OFF, informative messages are printed). Users may write their own RECLAIM P-EXEC if they wish, subject to the following restrictions:

1- When first called (&0=START), no P-EDIT command may be issued and the normal protective stack buffer is not made.

2- In both cases, nothing may be done to alter the P-EDIT, CMS or CP environment in a way that might interfere with the macro that might be in the middle of execution.

3- When called the second time (&0=DELAYED), any change is remembered as part of the command being executed when P-EDIT ran out of available storage and is subject to being undone by a subsequent UNDO command.

4- Keep me informed (KRUSKAL(YKTVMV)) that this non-standard RECLAIM P-EXEC is in use.

REMEMBERED COMMANDS see MEMORY

RESOLUTION

When P-EDIT gets a command, it tries to interpret it in different ways in turn until it succeeds. Simple interpretations are as a built-in command or a macro. Some interpretations were provided as a service to the SIMEDIT macro, and are probably not generally useful. Users may redefine the resolution order for the commands they enter, macros always get a standard order, although they can modify it. The kinds of resolution are:

Command: Try to interpret command as a built-in command using the built-in abbreviations if necessary. If found, do it; if not, try the next interpretation.

Macro: Try to interpret command as a macro, a file with type of 'P-MACRO' (NUMBER P-EXEC for numbers), using the normal CMS order for searching disks. If found, use EXEC2 to do it; if not, try the next interpretation.

Synonym: Try to find command name in the list of synonyms defined by the SYNONYM command. If found, try the next interpretation using the value found as the command name; if not, try the next interpretation with the original command name.

Command+: Try to interpret the command as a built-in command using the built-in abbreviations if necessary. If found, do it; if not, try the next interpretation. If the execution of the command detects a syntax error (return code of -1 for macros), try the next interpretation.

Macro+: Try to interpret the command as a macro If found, do it; if not, try the next interpretation. If the execution of the command detects a syntax error (return code of -1 for macros), try the next interpretation.

Error: If resolution has gotten this far, call the EXEC2 file ERROR P-EXEC if there is one.

ones provided primarily for SIMEDIT

Input: If the previous interpretations have failed, insert the line after the current line.

Uninput: If the previous interpretations have failed, print 'Rejected input' (return code of 7 for macros).

Rejectmacro: Try to interpret the command as a macro. If it is, print 'Rejected macro' (return code of 6 for macros); if not, try the next interpretation.

If none of the attempts at resolution are successful, the command is not valid (return code of -1 for macros). When entered in commands, the names of interpretations can be abbreviated as short as their first letter (except COMMAND+ and MACRO+, which must have the '+' (as in 'C+').

Examples of resolution orders:

SYNONYM COMMAND MACRO	Default for user entered commands
COMMAND MACRO	Default for macros
SYNONYM COMMAND+ MACRO	Permits a macro to be written that extends a built-in command in an upward compatible fashion and still use the built-in when possible

related documentation

COMMAND	RESOLVEORDER	(Change resolution order)
COMMAND	RESOLVE	(See how command resolves)
COMMAND	READ	(Temporarily change resolution order)
CONCEPT	SYNTAX	(Syntax errors)
CONCEPT	MACROS	(P-EDIT macro facility)
EXEC	ERROR	(May be used to handle syntax errors)
MACRO	SIMEDIT	(Permit commands and input to be intermixed)

RESOLUTION ORDER see RESOLUTION

RESOLVE ORDER see RESOLUTION

REVERSE see DIRECTION

RING COMMANDS

MACRO	BURN	(Stop editing current file and erases it on disk)
MACRO	CANCEL	(Leave P-EDIT with no protection for unsaved files)
MACRO	COMPARE	(Generate a file showing differences between two files)
MACRO	CONSOLE	(Edit console spool file)
MACRO	CPLIST	(Edit a list of spool files)
MACRO	DISKS	(Edit file showing accessed disks)
COMMAND	DROP	(Stop editing a file)
COMMAND	EDIT	(Read and edit a file)
MACRO	FILELIST	(Generate a list of files on disk)
MACRO	INSTANCE	(Generate an instance of a FORM file)
MACRO	OUT	(Stop editing all saved files)
MACRO	P-EDIT	(Resume editing or read file if not in memory)
MACRO	P-LIST	(Read a group of files for editing)
MACRO	QUIT	(Stop editing current file)
MACRO	RECEDIT	(Start editing file recursively)
MACRO	SEARCH	(Execute command over each of several files)
COMMAND	UNGO	(Will reincarnate previous file if necessary)

SCOPE see SCREEN FORMAT

SCREEN see SCREEN FORMAT

SCREEN FORMAT

When a 3270 device is being used and SCOPE is ON, the 24 lines of the screen are used as follows:

Line 1	Header Line	The following are adjusted left, center and right:
	WHO	Who is in control and level number if not 1
	WHERE	Information about the current line
	WHAT	Information about the current file
Lines 3-9	Previous lines	As many line before the current as will fit
Lines 10-c	Current line	Up to 850 characters of the current line
Lines c-m	Following lines	As many line after current as will fit
Lines m-21	Message Area	As many as 9 messages + extra scale line
Line 22	Scale Line	Tick marks to help typing fixed format lines
Lines 23-24	Input Area	130 character area to input commands and text

See: COMMAND SCOPE

header line

WHO field: The built-in values for this field are 'Edit:' (normal), 'Browse:' (BROWSE mode), 'Input:', 'Reverse input:', 'Replace:', 'Reverse replace:' (all various flavors of INPUT mode), 'Change:', 'Find:', 'Superimpose:' and Overlay:' (when the corresponding command is waiting to read its argument). In addition many macros set this field (the ':' is a conventional). In addition, if the editing level is not 1, to the above is added 'level' followed by the number ('Edit: level 2').

See: COMMAND HEADVARS	(How macros set the WHO field)
COMMAND READ	(How the level changes)

WHAT field: This field gives information about the current line. Normally it just has the position of the line in the physical file and the size of the file ('Line 25 of 50'). However the following exceptional conditions are noted: If the current line is a dummy line (status: TOP, EOF or DUMMY), the previous line is described for DUMMY and EOF and the next for TOP ('After line 25 of 50'). Otherwise, if the line is unfixed (status: UNFIXED, HIDDEN or EXCLUDED), that is noted ('Hidden line 26 of 50'). In addition, if more than 80 characters of the current line are displayed or WINDOW does not start at column 1 (and some characters of the line are displayed), the columns that are displayed are noted ('Line 25 of 50, cols 3-70').

See: CONCEPT CURRENT LINE
     CONCEPT STATUS
     COMMAND WINDOW

WHO&WHERE field: In a few cases P-EDIT uses these two fields as one, namely, while a file is being written to disk the name of the file follows 'Filing as:' or 'Appending as:'. In addition, certain macros set this double field.

See: COMMAND SAVE
     COMMAND APPEND
     COMMAND HEAD

WHAT field: This field describes the current file being edited. It consists of the file number and the file name, type and mode ('2. PROFILE P-EXEC A2').

previous, current and following lines

Lines 3 to the message area (or 21 if none) contain as much of the file around the current line as possible. No more than 850 characters of any line are displayed. Because it takes an extra character to switch from bright to normal display (and back again) some lines might be missing their last character (P-EDIT tries to find a blank character to use for this purpose).

See: COMMAND WINDOW	(Controls which part of the lines are displayed)
COMMAND SCOPE	(Controls how to display long lines and bad characters)
COMMAND HILITE	(Controls treatment of current and unfixed lines)

message area

All messages issued by P-EDIT are placed into the message area, which changes in size as needed (maximum of 10 lines). If there are any messages, the scale line is repeated above the messages. If there are more messages waiting to be displayed, the lower right-hand corner of the screen will say 'MORE...'. Pushing CANCEL will display them.

See: COMMAND SAY   (Used by macros to issue messages)

scale line

A scale line is displayed just above the input area to help the user know what column he is typing into. For technical reasons, it can be only 79 characters long. It basically consists of a tick mark in every column. However, every column whose column number is a multiple of ten has the ten's digit of its number and midway between these columns there is a longer tick mark:

```
''''!''''1''''!''''2''''!''''3''''!''''4''''!''''5''''!''''6''''!''''7''''|''''
```

See: MACRO SCALE   (Inserts a similar scale line into file)

Special characters are inserted into the scale line to inform the user of the settings of ZONE, TRUNC and WINDOW. Often these characters cannot be shown, since the column is not represented on the scale line. These special characters are:

>	First column of ZONE
<	Last column of ZONE
X	First and last column of ZONE
/	TRUNC column
%	TRUNC column when it is the same as last column of ZONE
|	Last column of WINDOW

See: COMMAND ZONE	(Controls zone for file searches and modifications)
COMMAND TRUNC	(Controls truncation for file insertions)
COMMAND WINDOW	(Controls part of lines displayed)

input area

The input area is where commands for P-EDIT are entered and text to be inputted is prepared. An extensive local editing feature exist on the 3270 intended for correcting mi,takes while typing. This is exploited by a number of commands that place information into the input area for the user to modify and re-enter just as though it had entirely originated with him. If the alarm on the terminal is sounded, the input area is cleared.

See: COMMAND ALARM	(Can be used to turn off the alarm on warning messages)

SETUP see INITIALIZATION

STATUS

In P-EDIT, the Boolean expression of a line and the Boolean expressions established as the current masks together classify the line into one of seven classes. This is called the status of the line. By using the STATUS command, the status of a line can be printed (stacked). The STATUS command can also be used to go to a line with a particular status. When the PRINT command prints a line with a status other than FIXED, its status is printed with it.

See: CONCEPT PARAMETRIC FILES	
COMMAND STATUS	(Print or find for status)

status in non-parametric files

FIXED: A line which occurs in all versions of the file consistent with the current masks. Typically most lines have the status FIXED.

DUMMY: A line which is not in the file. These lines are not written to disk by the SAVE command. When a line is deleted, it assumes the status DUMMY. It remains in the image of the file until no tag refers to it, it is not the current line, and the storage it occupies is otherwise needed. The only ways a line with the status DUMMY can be noticed is by the STATUS, GO or UNGO commands.

TOP: A dummy line placed before the first line of a file.

EOF: A dummy line placed after the last line of a file.

status in parametric files

UNFIXED: A line which occurs in some, but not all, of the versions that are consistent with the current masks is called "unfixed". The lines having status UNFIXED are unfixed lines which are in the one version that P-EDIT displays, called the view. These lines are highlighted in some fashion.
    See: COMMAND HILITE   (Control how UNFIXED lines are highlighted)

HIDDEN: An unfixed line which is not in the view. For most purposes these lines are treated the same as lines with the status UNFIXED. A CHANGE command will, for example, change lines with status FIXED, UNFIXED, or HIDDEN. Commands which go to new lines (such as NEXT and LOCATE), can easily go to a hidden line. When this happens, the line is displayed as blanks and the header on the screen will warn of this anomaly. The text of the line can still be seen by using the PRINT command or changing the view.
    See: MACRO UNHIDE   (Change view to see current line)

EXCLUDED: A line which is in some version of the file, but is not in any version that is consistent with the current masks.

table that organizes preceding information

Type of Line (based on Line's Boolean expression)	Real File	Logical File	Viewed File	Line Status
Proved by the masks		fixed	normal	FIXED
Independent of masks & consistent with view			hilited	UNFIXED
Independent of masks & inconsistent with view	real	unfixed		HIDDEN
Inconsistent with masks & not FALSE				EXCLUDED
Line is top dummy line		excluded	hidden	TOP
Line is end-of-file dummy line	dummy			EOF
Line has been deleted				DUMMY

STORAGE see RECLAMATION

SYNTAX

P-EDIT commands consist of edit tokens (such as the command names, numeric arguments and options), and text (to be searched for or inserted in the file).

Edit tokens: Most of the edit tokens fall in one of three categories: alphabetic tokens consisting of one or more letters, numeric tokens consisting of one or more digits, and non-alphanumeric tokens consisting of a single non-alphanumeric character. A token is terminated by a blank or when the next character cannot be added to it. Often, therefore, tokens may be juxtaposed without introducing ambiguity, as in the following examples:

```
print2
change/a leaf out of/P-EDIT's book/10g
zone**
```

Exceptions are hexadecimal arguments for ALTER (letters are digits), command names beginning with 'P-' ('-' is alphabetic), and tokens that are part of CMS file names (all but '*', '(' and ')' are alphabetic).

Text: Special treatment may be given to text which contains back-spaces or tabs.

See: COMMAND IMAGE   (Controls handling of back-space and tab)

When a line of text follows a command name, in the same line of input, the text is deemed to start at the character which terminates the command name, if this is not a blank, or at the next character if it is a blank. Thus the first two of the following commands insert an '*' in column 1 and the last inserts it in column 2:

```
INSERT*
INSERT *
INSERT  *
```

Users should be warned that CMS does not support more than 130 physical characters (keystrokes) in one physical line of input.

All commands are carefully checked for proper syntax. Syntax errors are detected prior to the command having any effect (certain compound commands (such as GROUP) are an exeption to this). If not issued by a macro (sub-command) the warning message '?Edit: ' is printed followed by the bad command. Macros receive a return code of -1.

syntax description language

Example of syntax description for the MOVE command:

```
<-> MOve <<tag1>-tag2> <To tag3>
    COpy       n          Up m
               *          Up *
                         Down m
                         Down *
                          Here
```

<->: The brackets indicate that the field is optional. This particular optional field is shown for all commands that respond to direction.

MOve: The first letter is capitalized to indicate that the field being shown is to be interpreted literally, as itself. When used, the field can be in lower-case. The last two letters are in lower-case to indicate that they are optional.

COpy: This is below MOve to show that it is an alternative for it.

<<tag1>-tag2>: The first letter is lower-case to indicate that the field being shown is to be interpreted as a representation of the field, a name, not literally. These names are usually chosen to indicate the meaning of the field, in this case a tag. This field is an example of an optional field that contains a further optional field.

n: This also the name of a field; when used, a number would be coded to specify the number of lines to be moved. It is centered below the previous field to indicate that it is an alternative to the entire field, <<tag1>-tag2>, not just one part of it, say <tag1>.

...: Not shown above, but in the following example, an ellipses is sometimes used to indicate any number of occurrences. In the following, there must be at least one string. A '<' before 'string1' would have indicated that not having any strings were permitted.

```
<-> LAND /string1</string2 ... ></>
```

TAGS

Tags are a facility which allows a line to be given a name. Once a line has a name, it can be made the current line and can specify the limiting line of a range in most commands which accept a line count:

```
GO tag
CHANGE /apple/pair/ tag
```

Tags are are not private to a single file, each one must be unique over all files in the ring. Therefore they are useful to hold multiple positions in a set of files. A GO command which specifies a tag in a file, other than the current one, will redefine the current file as well as the current line within it.

When tags are used to specify the limit of a range, no check is made to see how or whether it is defined. Thus, if it is attached to a line in another file, to a line does not follow the current line, or to no line at all, it will be equivalent to specifying a limit of '*'.

The syntax of tag names is the same as P-EDIT variable names: upper-case alphanumeric words separated by '.', beginning with a letter, with blanks permitted between words. Examples: BEGINNING and DEFAULT.FIRST.TAG

See: COMMAND	TAG	(Set tag for current line)
COMMAND	UNTAG	(Clear the specified tag)
COMMAND	GO	(Go to the line associated with the specified tag)
COMMAND	TAGS	(Print tags on lines in the specified range)
MACRO	FINDTAG	(Go to the next occurence of tag fitting pattern)
MACRO	RANGE	(Set default tags for other macros)

UNDO see MEMORY

VARIABLES

P-EDIT variables are a facility which allows a name to be given any string as a value. They are primarily used by P-EDIT macros to communicate between themselves.

Variables are global to all files in the ring. When a new level of editing is entered, the variables are initialized to their previous value. When the level exits, they are restored.

The syntax of variable names is the same as tag names: upper-case alphanumeric words separated by '.' beginning with a letter with blanks permitted between words. Examples: BEGINNING and COMMAND.X

See: COMMAND	STORE	(Define a variable)
COMMAND	UNSTORE	(Clear a variable)
COMMAND	FETCH	(Get the value of a variable)
COMMAND	VARS	(Get list of defined variables)
MACRO	SHOWVARS	(Edit a file showing variables and values)
COMMAND	HEADVARS	(Declare variables to control screen header)

VERSIONS see PARAMETRIC FILES

VIEW see PARAMETRIC FILES

VIEWED FILE see PARAMETRIC FILES

*

```
* <text>
```

Comment; does nothing except to be added to the list of remembered commands.

; see GROUP

¬ see LOCATE

- see REVERSE

/ see LOCATE

?

```
? <n <m>>
    L
    LL
```

Two numbers specified: Prints (stacks) the remembered commands from the first to the second. Remembered commands are numbered from 1 (the previous command) on. They are printed (stacked) as: ? n = command

One number specified: Prints (stacks) that remembered command. If SCOPE is ON, it is placed into the input area where local editing can modify it and pushing ENTER will execute (or re-execute) the command.

No argument specified: Same as: ? 1

L specified: Same as: ? 1 9

LL specified: Same as: ? 1 99999999

related documentation

CONCEPT MEMORY	(How P-EDIT remembers commands)
COMMAND RETRIEVE	(Similar command with additional flexibility)
COMMAND SCOPE	(Controls printing of one command)

```
" <command>
```

Stacks (FIFO) the last edit command and then executes the specified command. This normally has the effect of redoing the previous command after the optional additional command. Certain commands don't count in determining the last command: null commands, the ? command, the " command issued with no argument, and commands hidden with COVERT. Some examples:

```
     c/x/y/
     "n2
```

This changes 'x' to 'y' in the current line, and then repeats the operation two lines down.

```
     This line was intended as input but caused an error
     "i
```

The line typed in under the belief that P-EDIT was in INPUT mode is inserted in the file and INPUT mode is established.

ALARM

```
ALARM <ON>
       OFF
```

Argument specified: Sets the mode of whether the audible alarm (on devices which are so equipped) should be sounded or not when an error message is printed. The initial setting is ON.

No argument specified: Prints (stacks) the current setting.

See: COMMAND SHOUT	(Used by macros to print warning messages)

ALTER

```
<-> ALter char1 char2 <tag <p>>
                        *    *
                        n    G
```

Alters the first (p-th) occurence of the first character specified to the second on each line of the specified range. The last line scanned becomes the new current line. Normally only the first (p-th) occurrence in a line is altered; to alter all occurrences, add '*' (or, equivalently, 'G' for "Global").

A character may be specified normally or in hexidecimal as two contigous digits such as '05' or '7B'. No translation to upper-case is ever performed on bytes in the file specified as hexadecimal.

If the first character is not found, 'Not found' is reported, but the new current line will be the same as if the command had been successful.

related documentation

COMMAND CASE	(Controls the handling of lower-case letters)
COMMAND DESERIALIZE	(Controls serialization field to be cleared)
COMMAND ZONE	(Controls the part of the lines that are considered)
COMMAND CHANGE	(Similar command for strings of characters)

APPEND

```
APPEND  <fn <ft <fm <format> <width>>>>
         *   *   *
```

Adds the current file to the end of the specified file on disk (if the file doesn't exist, the current file becomes a new file on disk). Appends the file to existing one on disk or creates new one if none. Missing arguments (or '*' for the parts of the file name) are taken to be the corresponding values for the current file.

The file is written on the disk specified by the first letter of fm. Each line written is padded with trailing blanks to the width if the format is F or FP. If the format is VB or V, the lines are written with one or no trailing blanks. V, VB, VP, VBP, and parametric FP format files appear as V format on disk, and the maximum line length of the file on disk may be less than the width.

If an error occurs during the operation, the file will, in general, have been modified.

related documentation

COMMAND NAME	(Controls the default file name, type and mode)
COMMAND FORMAT	(Controls the default format)
COMMAND WIDTH	(Controls the default width)
COMMAND SAVE	(Similar command that rewrites existing file)

ATBASE

```
ATBase
```

Declares that the current masks define the "base" of the file, the normal masks for editing the file.

It can happen that a user sets a mask and then forgets he has done so. This would result in all further editing being done in a more restrictive environment than he thinks. To make this less likely, P-EDIT has the notion of a "base" Boolean expression. Whenever the AND of the current masks is different than the base, the heading line on the screen is displayed bright.

The initial base is TRUE, but certain parametric file types have P-SETUP files that initialize a mask (such as TIME) and they would establish that mask value as the base.

See: CONCEPT PARAMETRIC FILE
     CONCEPT INITIALIZATION

AVOID see LOCATE

BLANK

```
<blanks>
```

If a command consisting of only one or more blanks is not resolved, the normal case, it will be totally ignored. It could be resolved by INPUT and the like.

The null command, consisting of no characters, normally does nothing except refresh the scope. However if SCOPE is set OFF and there are no stacked lines, the information normally displayed as the left most field in the screen header line (such as 'Edit:') will be printed. In a parametric file if the masks are not at their base (defined by ATBASE), the printed information will be preceded by 'MASKS...'; were SCOPE ON, this warning would be conveyed by displaying the screen header bright. In any case, the null command is not included in the remembered commands.

```
See: CONCEPT RESOLUTION      (Way that resolution might insert a blank line)
     COMMAND SCOPE           (Controls whether SCOPE is ON or OFF)
     COMMAND ATBASE          (Controls definition of base file)
```

BOTTOM see TOP

BROWSE

```
BRowse <OFF>
        ON
```

Argument specified: Sets BROWSE mode accordingly.

No argument specified: Prints (stacks) BROWSE mode of current file.

If the BROWSE mode of a file is on, it may not be modified or saved on disk. A file is put into BROWSE mode if a file with the same name is already being edited, when editing of it begins. This is dangerous situation since one file can easily overwrite the other. This situation can occur only by using the built-in EDIT command. It is more common to use the macro, P-EDIT (with the abbreviation 'E'), which prevents this problem by resuming the file if it is already being edited.

```
See: MACRO P-EDIT      (Normally used to start editing a file)
```

CALLER see READ

CASE

```
CASe  <U <U>>
       M  M
```

Argument specified: Sets "input translation" mode and, if specified, "searching translation" mode accordingly.

No argument specified: Prints (stacks) a line with two words, the input and searching translation for the current file.

details

Translation is coded as 'U' ("Upper") to translate lower-case letters to upper-case or 'M' ("Mixed") for no translation. The input translation controls whether strings that are to be placed into the file should be translated (such as the argument of INSERT or the second argument of CHANGE). The searching translation controls whether strings in the file are to be temporarily translated when compared argument strings (such as the argument of LOCATE or the first argument of CHANGE). The argument that is used for a search is translated to upper-case if either the input or searching translation is 'U'.

The initial translation of a file is 'U M', but P-SETUP files for many file types (such as SCRIPT) change the translation to 'M M'.

CHANGE

```
<-> Change </string1</<string2</ <tag <p>>>>>>
                                      *   *
                                      n   G
```

Changes the first (p-th) occurrence of first specified string to the second specified string on each line of the specified range. Changed lines are recanonicalized if IMAGE is CANON. The last line scanned becomes the new current line. Normally only the first (p-th) occurrence in a line is changed; to change all occurrences, add '*' (or, equivalently, 'G' for "Global").

The delimiter (shown here as '/') may be any non-blank character not in the strings and is optional if at the end of the command. Missing strings are taken as null. Thus, if the first string is null, the second string is imbeded at the beginning of each line, unless a global change is specified, in which case the line will be filled with repeated copies of the second string.

If the first string is not found, 'Not found' is reported, but the new current line will be the same as if the command had been successful.

related documentation

COMMAND CASE	(Controls the handling of lower-case letters)
COMMAND DESERIALIZE	(Controls serialization field to be cleared)
COMMAND IMAGE	(Controls the handling of back-space and tab)
COMMAND ZONE	(Controls the part of the lines considered)
COMMAND ALTER	(Similar for hexidecimal specification)
RESOLVE CMC	(Similar for mixed-case change)
RESOLVE CMF	(Similar for changing a string unique to file)
MACRO CUS	(Similar for changing a string unique on the screen)
MACRO INCHANGE	(Similar but permits interactive user control)

description of CHANGE with no arguments

If a 3270 with a keyboard is used, the CHANGE command may be issued with no arguments. In this case the text of the current line's ZONE is put into the input area where it can be modified by local editing. When ENTER is pressed the possibly modified characters replace the ZONE of the line, unless the input area is entirely "nulls", as would be generated by ERASE INPUT or CLEAR.

Note -- if there are unprintable characters in the line or characters which are not translated back to themselves by the input and input area translation tables (in succession), the command will be rejected.

See: COMMAND TRANSINX (Controls input translation tables)

CMS

```
CMS
```

Enter CMS SUBSET. In this mode it is possible to execute those CMS commands which do not require main storage. These include RENAME, TYPE, CP, DISK, ERASE, EXAMINE, EXEC, PUNCH, FILELIST, QUERY, ACCESS, READCARD, PRINT, RDR, RELEASE, SYNONYM, STATE and SET. Any attempt to execute an invalid CMS command or one which requires main storage will provoke an error message. In general, the LOAD command (and its relatives such as LOADMOD) should be avoided.

To resume editing, enter the special CMS SUBSET command, RETURN.

```
See: MACRO $        (Execute specified CMS commsnd)
     MACRO CP       (Execute specified CP command)
```

discussion of translation tables and PF-keys

The editor temporarily sets the state of the input and output translate tables as well as the PF-keys, to the state they were in before the editor was ever invoked. Upon returning from CMS SUBSET, they are reset to the state they were in immediately before entering CMS SUBSET. Thus, to change their settings for the editor, use one of the previously mentioned macros.

```
See: MACRO   TRANSIN     (Controls translation of characters on input)
     MACRO   TRANSOUT    (Controls translation of characters on output)
     COMMAND PFKEY       (Controls settings of PF-keys)
```

COLUMN

```
COLumn /<string</>>
```

Searches the current line for the specified string and prints (stacks) the starting column number. If string is not found, 'Not found' is reported. Primarily for use by macros.

The delimiter (shown here as '/') may be any non-blank character not in the string.

```
See: COMMAND CASE     (Controls the handling of lower-case letters)
     COMMAND ZONE     (Controls the part of the line searched)
```

Note -- this is different than EDITOR which printed (stacked) '0' if the string was not found.

COMMAND

```
<-> COMMAND command
```

Executes the specified command as a built-in command regardless of the current resolution order. Primarily for use by macros.

```
See: CONCEPT RESOLUTION
     COMMAND MACRO           (Forces interpretation as a macro)
```

COPYFROM see MOVEFROM

COVERT

```
COVert command
```

Executes the specified command without remembering it. Thus it will not be seen by the ? or " commands. For purposes of the UNDO and UNGO commands, its effects will be absorbed into those of the previous command.

Primarily useful for assignment to PF-keys and by macros that use UNDO and UNGO.

See: CONCEPT MEMORY

DELETE

```
<-> Delete <tag>
            n
            *
```

Deletes the lines in the specified range. The new current line is the one following the deleted lines. Therefore DELETE followed by DELETE has the same effect as DELETE 2.

```
See: MACRO DOTHERS      (Deletes all lines not meeting some criteria)
     MACRO DTOP         (Deletes all lines from the previous to the TOP)
```

reincarnation of deleted lines

Deleted lines remain in memory with the status, DUMMY. This will normally not be apparent since these lines seem not to exist for almost all purposes (they are never written on disk, for example). However, a few commands can force the one of these DUMMY lines to become the current line. Should this happen, the line on the display reserved for the current line will be blank and the header line of the screen will be similar to what it is when the dummy EOF line is the current line.

```
See: COMMAND GO         (Can go to a tagged deleted line)
     COMMAND UNGO       (Can go to a deleted old current line)
     MACRO   MOVE       (Can leave the current line of another file deleted)
     COMMAND STATUS     (Can find a DUMMY line)
```

DELIMSTRING see TOKEN

DESERIALIZE

```
DESERialize  <ON p q>
              OFF
```

Argument specified: Turns on deserialization for columns p through q, or turn it off. When deserialization is on, all commands which modify, insert or move lines blank out the serial number field (columns p through q) of those lines.

No argument specified: Prints (stacks) the current settings if no argument is

```
See: MACRO RESERIAL     (ReseriaLhe lizes lines in range)
     MACRO UNSERIAL     (Clears serialization of lines in range)
given.
```

DIRECTION see REVERSE

DOWN see NEXT

DROP

```
DRop <fnumber>
```

Removes the specified file (or current file) from the ring of files being edited. If the current file is dropped, the previous file in the ring becomes the current file. If there is no previous file, the last file in the ring becomes the current file.

Primarily for use by macros.

See: MACRO QUIT   (Normal way to stop editing a file)

EDIT

```
Edit <fn <ft <fm <format <width>>>>> <( <Dummy> <Nosetup> <DARK> <Quiet> <)>>
      *   *   *
```

No argument specified: Prints (stacks) one line describing each file being edited followed by a null line.

File specified: Edits the specified CMS file. If the file is already being edited, BROWSE mode is set on and a warning is printed. If the file does not exist or the file is a dummy file (see below) an empty file is edited.

The standard SETUP P-EXEC defines a synonym for the macro, P-EDIT, that requires at least 'ED' to be entered to specify the built-in command, EDIT. This macro is the normal way to start editing a file.

See: CONCEPT INITIALIZATION	(Initialization of files by file type)
COMMAND FORMAT	(Format of a file)
COMMAND WIDTH	(Width of a file)

details of arguments

fn: The file name of the file. If '*' or missing, the file will be a dummy file (see below).

ft: The file type of the file. If '*' or missing, the file type of the current file is assumed.

fm: The file mode of the file. If '*' or missing, the first disk on which the file is found (normal CMS search order) is assumed unless the file is a dummy file (in which case it will be '*'). If the mode number is missing, '1' will be assumed.

format: The format the file should have. If missing, the file will be 'V' or 'F' depending on its format on disk.

width: The width the file should have. If the file has a line longer than the specified width, the reading of the file will be terminated at that point and an error message will be printed.

details of options

DUMMY: The file is a dummy file and will not be read from disk. The file mode must be missing or '*' if this option is specified. When a file is a dummy file (DUMMY option or file name missing or '*') the initialization file executed will have its sixth argument 'YES' (normally 'NO').

NOSETUP: No initialization file will be executed to initialize the file.

DARK: The initialization file executed will have its seventh argument 'NO' (normally 'YES'). This argument is ignored by the standard initialization files.

QUIET: The initialization file executed will have its eighth argument 'NO' (normally 'YES'). This argument is ignored by the standard initialization files.

EDITING

```
EDITING <fn <ft <fm>>>
         *   *   *
```

Prints (stacks) a line containing the file numbers of each file being edited whose name matches the pattern specified. A absent argument or '*' will match anything. Primarily for use by macros.

EDITMODE

```
EDITMode <Text>
          Boolean
```

Argument specified: Sets the edit mode accordingly.

No argument specified: Prints (stacks) the edit mode of the current file.

Normally the edit mode is TEXT. When the edit mode is BOOLEAN, Boolean expressions are treated as though they were the text of the line. Used primarily by macros.

Warning -- All sorts of file characteristics such as WIDTH and CASE that made sense in TEXT mode will act strangely in BOOLEAN mode. Use of the BOOL command is recommended for this reason.

See: CONCEPT PARAMETRIC FILES	
MACRO BOOL	(Executes command in BOOLEAN edit mode)

ERROR

ERROR

This command is equivalent to a command with a syntax error. It can be used to, effectively, clear the definition of the X or Y command.

See: CONCEPT SYNTAX	(Treatment of syntax errors)
COMMAND X	(Provides a short name for specified command)

EXCLUDE see FIX

EXCLUDEVIEW see FIXVIEW

EXIT see READ

EXVIEW see FIXVIEW

FCOUNT

FCOUNT

Prints (stacks) the number of files currently being edited.

FNUMber

Prints (stacks) a line containing the file number of the current file and the file number of the file whose original control is currently being used (normally the same). Used primarily by macros.

File numbers are assigned sequentially whenever the editing of a file begins, starting from 1. These serve to provide a unique identification for each file. This is necessary because P-EDIT permits multiple copies of files with the same CMS file name.

See: COMMAND XFERCONTROL (Description of controlling files)

FETCH

FETch name

Prints (stacks) the current value of the specified variable.

STOre name <value>

Value specified: Defines the specified value as the value of the specified variable. In this form the value may not be null or begin with a blank or a '.'.

No value specified: The value is read separately and the operation is otherwise the same. If the value is null, acts the same as UNSTORE.

UNSTORE name

Makes the specified variable no longer defined.

See: CONCEPT VARIABLES

FIND

<-> Find <line>

Line specified: Searches the file, from the next line, examining only those characters in each line which correspond in position to the non-blank characters in the specified line. The first line in which these characters match those given becomes the new current line. If none is found, 'Not found' is reported and the current line becomes the dummy EOF line. If line is null or blank, the search will be successful on the first line examined. If the current line is the dummy EOF line when this command is issued, the search starts from the dummy TOP line.

No line specified: The line is read separately and the operation is otherwise the same.

This command usually operates faster than LOCATE.

related documentation

COMMAND CASE	(Controls the handling of lower-case letters)
COMMAND IMAGE	(Controls the handling of back-space and tab)
COMMAND ZONE	(Controls the part of the line examined)
COMMAND LOCATE	(Similar command that does not consider position)

FIX

```
FIX <mask> <tag>
           n
           *
```

Further restricts the specified mask by the minimum Boolean expression which will cause the lines in the specified range to be fixed.

```
EXclude <mask> <tag>
               n
               *
```

Further restricts the specified mask by the minimum Boolean expression which will cause the lines in the specified range to be excluded.

See: COMMAND FIXVIEW (Similar commands for lines in the view)
     CONCEPT PARAMETRIC FILES

FIXVIEW

```
<-> FIXView <mask> <tag>
                   n
                   *
```

Further restricts the specified mask by the minimum Boolean expression which will cause those lines in the specified range that are in the current view to be fixed. If the range is specifed by a number, it refers to the lines in the logical file as usual, not the viewed file.

```
<-> EXCLUDEView <mask> <tag>
    EXView             n
                       *
```

Further restricts the specified mask by the minimum Boolean expression which will cause those lines in the specified range that are in the current view to be excluded.

See: COMMAND FIX (Similar commands for all lines in range)
     CONCEPT PARAMETRIC FILE

FM see NAME

FN see NAME

FNUMBER see FCOUNT

FORMAT

```
FORMat  <F<P>>
RECform V<B><P>
```

Format specified: Sets the record format (for subsequent SAVE commands) to 'F' (fixed) or 'V'. Usually a 'V' format file will occupy less disk space than the corresponding 'F' format, since in the 'V' case trailing blanks are stripped automatically from each line before it is written out.

No format specified: Prints (stacks) the format of the current file.

The special format VB will append one trailing blank per line when the file is written. This is necessary for 'V' format PLI or LISP files.

If the last letter of the format specification is 'P', the file becomes parametric. Once a file is parametric, it will remain so.

```
See: COMMAND SAVE                  (Saves the current file on disk)
     CONCEPT PARAMETRIC FILE
```

FREE see RECLAIM

FT see NAME

GETFILE

```
GETfile fn <ft <fm <m <n (<Quiet<)>>>>>
            *   *   *  *  *
```

Inserts some or all of the given file, following the current line. Any part of the file name that is missing or is an '*' will be taken to be the same as the current file. The part of the file which is inserted runs from line m, for n lines, or to its end. If the starting line is missing or is an '*', it will be taken to be '1'; likewise, the number of lines will be taken to be large enough to read the remainder of the file. The message 'k lines read', where k is the number of lines actually inserted, is printed (stacked) unless the QUIET option is specified. The last line inserted becomes the new current line.

If the format of the current file is parametric (VP, FP or VBP), GETFILE will look for Boolean expressions on the incoming lines.

related documentation

```
COMMAND DESERIALIZE         (Controls serialization field to be cleared)
COMMAND WIDTH               (Controls the widest line that may be inserted)
MACRO   GETTAIL             (Gets the last part of specified file)
MACRO   GETFILES            (Gets all files for pattern specified)
CONCEPT PARAMETRIC FILES
```

GO see TAG

GROUP

```
<-> <GRoup> ;command1 <;command2 ... >>
```

The commands, delimited by the first character, are executed in turn until an error or other exception is found. The delimiter, shown as a ';', can be an non-blank character not in the commands. If it is a ';', the command name, GROUP, can be ommitted.

The effect of reverse mode is to execute each command in reverse mode (or actually in the opposite mode that the command itself specifies).

```
See: CONCEPT DIRECTION
     MACRO   IF              (Permits conditional execution of commands)
```

HEAD

```
HEAD <string>
```

Temporarily replace the display header with the specified string. The maximum length of the header is 40 bytes. If issued with no arguments, the temporary header is read from the console stack. The temporary header applies only to the next display update. Mostly useful in macros, before issuing the SCOPE command to refresh the screen.

```
See: COMMAND SCOPE          (Used to refresh the screen)
     COMMAND HEADVARS       (Controls the left, "WHO", part of the header)
SCOPE REFRESH.
```

HEADVARS

```
HEADVars <name1 <name2 ... >>
```

Defines the variables that are to provide the WHO field of the header line. The value of the earliest of these that has a value will be used for the header, unless its value is 'FILE#' or 'CONTROL#. In that case, the variable whose name is the concatenation of of the original name, '.' and the file or control number of the current file (as given by the FNUMBER command) will be looked up and used if defined.

If none of these are defined, the header will say 'Edit:' or 'Browse:' depending on BROWSE mode. Other built-in values (such as 'Input:') take priority over this process.

Used primarily by SETUP P-EXEC.

```
See: CONCEPT SCREEN FORMAT      (All about the header lines)
     CONCEPT VARIABLES          (All about P-EDIT variables)
     CONCEPT INITIALIZATION     (HEADVARS usually issued by SETUP P-EXEC)
     COMMAND STORE              (Sets the value of a variable)
```

HILITE

```
HILite <char>
        OFF
```

Character specified: Sets the HILITE mode of the current file to that character, thus freeing up the brightening feature of the 3270 terminal to highlight the current line. Lines with the status of UNFIXED will be highlighted by changing all blanks on the line to the HILITE character.

OFF specified: Sets HILITE OFF, thus the current line will not be highlighted and UNFIXED lines will be displayed bright.

No argument specified: Prints (stacks) the HILITE mode for the current file.

Note -- The only way to assure that all UNFIXED lines will be noticed is to have a HILITE character and set the window, WINDOW 1 78.

See: CONCEPT PARAMETRIC FILES
     COMMAND WINDOW

IMAGE

```
IMage  <ON>
       CANON
        OFF
```

Argument specified: Sets IMAGE mode of the current file accordingly.

No argument specified: Prints (stacks) IMAGE mode of current file.

See: CONCEPT IMAGING

INPUT see INSERT

INSERT

```
<-> Insert <line>
    INPut
```

Line specified: Inserts it in the file following the current line, and makes it the current line.

No line specified: Enter INPUT mode in which lines read from the terminal are put in the file following the current line. As each line is entered, it becomes the new current line. EDIT mode is restored by entering a null line. A blank line is inserted by typing at least one blank.

```
<-> Replace <line>
```

Same as INSERT except the current line is replaced by the first (or only) insertion.

related documentation

COMMAND CASE	(Controls the handling of lower-case letters)
COMMAND DESERIALIZE	(Controls serialization field to be cleared)
COMMAND IMAGE	(Controls the handling of back-space and tab)
COMMAND TRUNC	(Controls the longest line that can be inserted)
MACRO REINPUT	(Permits P-EDIT commands to be entered in INPUT mode)

LENGTH

LENgth

Prints (stacks) a line containing two numbers: the length of the current line up through the truncation column after discounting trailing blanks (working to the left from the truncation column), followed by the total length of the line (ignoring the truncation column). Primarily for use by macros.

See: COMMAND TRUNC   (Sets truncation column)

MAXLength

Prints (stacks) a line with two numbers: The length of the longest line in the logical files, followed by the length of the longest line in the physical file.

```
See: CONCEPT PARAMETRIC FILE
     COMMAND EDITMODE            (Meaning of "line" independent of EDITMODE)
```

LEVEL see READ

LINES

<-> LInes <n>

Line number specified: Positions the file at line number n counting from the top of the real file. If n is 0, the dummy TOP line becomes the current line; if n is larger than the file, the dummy EOF line becomes the current line (return code of 1 for macros).

No line number specified: Prints (stacks) a line with three numbers: the ordinal number of the current line in the real file, the number of lines in the real file, and the number of lines with (non-TRUE) Boolean expressions (0 for non-parametric files). If the current line is the dummy TOP line, 0 is reported as its number; if the dummy EOF line, a number one greater than the number of lines in the file is reported; if any other dummy line (status of DUMMY), the line number of the previous line (or 0 if none) is reported and a return code of 1 is provided macros.

See: CONCEPT PARAMETRIC FILES

LOCATE

```
<-> <Locate> /string</ <tag>>
                        n
                        *
```

Searches the logical file, from the next line over the specified range for the first line containing string, and makes it the new current line. If the count is 0 or the current line has the specified tag, it will fail. The delimiter, given here as '/', may be any non-blank character. If the opening delimiter is '/', the word LOCATE need not be given. If no range is specified, the entire remander of the file will be searched.

```
<-> <AVoid> ¬string<¬ <tag>>
                       n
                       *
```

Similar to LOCATE except that a line not containing the string is searched for and the string delimiter which makes AVOID optional is '¬', not '/'.

All lines contain the null string ('//').

If the current line is the dummy EOF line when the command is issued, searching starts from the dummy TOP line.

If the search is unsuccessful, 'Not found' is reported. If the current line is not now the dummy EOF line, a dummy line is generated which is after the last line searched and before the next, the dummy line becomes the current line.

related documentation

COMMAND CASE	(Controls handling of lower-case letters)
COMMAND ZONE	(Controls the part of the lines searched)
MACRO AVAND	(Go to next line missing one of specified strings)
MACRO AVOR	(Go to next line missing all specified strings)
COMMAND FIND	(Go to next line with charaters in same positions)
MACRO LAND	(Go to next line with all specified strings)
RESOLVE LMC	(Go to next line with specified mixed-case string)
MACRO LOR	(Go to next line one of specified strings)
MACRO LUF	(Go to line with specified string uniquely in file)
MACRO LUS	(Go to line with specified string uniquely on screen)

MACRO

```
<-> MACRO command
```

Executes the specified command as a macro regardless of the current resolution order.

```
<-> XMACRO fn <ft <fm>>
                *   *
```

Reads the arguments for the macro (including &0) separately and executes the specified EXEC2 file. If the file type is missing or '*', it will be taken as 'P-MACRO'. If the file mode is missing or '*', it will be taken as the first disk on which the file is found (the normal CMS order).

Primarily for use by macros.

```
See: CONCEPT RESOLUTION
     CONCEPT MACROS
     COMMAND COMMAND          (Forces interpretation as a built-in command)
```

MACROLINE

```
MACROLINe
```

Stacks the command that initiated the current macro being executed. This is primarily to allow macros to tell whether their arguments were separated from their name by a blank or not.

MAKE

```
<-> MAKE tag eqrel bool
         n
         *
```

Changes every occurrence of the specified equal relation to the specified Boolean expression in the Boolean expression associated with each line in the specified range. This is done in such a way that no new versions of the file can be created (e.g., ¬eqrel --> ¬bool).

See: CONCEPT PARAMETRIC FILES

examples -- TOP and no masks assumed

```
MAKE * PARM=YES TRUE                      Purge parameter, PARM
MAKE * VER=5 VER<=5                       Purge versions less than 5
MAKE * VER=4 4<=VER<6                     Purge version 5
MAKE * SEX=N FALSE                        Purge parameter value
MAKE * SEX=M SEX=M|SEX=N                  Make two values synonymous
MAKE * SEX=N SEX¬=M&SEX¬=F                Remove redundant value
MAKE * SEX=N NOSEX=YES                    Promote value to parameter
MAKE * SEX=M SEX=MALE                     Rename value
FOR 2 MAKE * P=1 P¬=1                     Remove all values but 1
FOR 2 MAKE * P=1 ¬MEMBER(P,(1,3,6))       Remove all values but 1, 3, & 6
```

MASK

```
MASK <mask> <(bool)>
```

Boolean expression specified: Resets the specified mask (or the current default mask) to the expression specified.

No Boolean expression specified: Prints (stacks) the current value of the specified masks (or if none the name of the current default mask).

```
UNMask <mask>
```

Clears the specified mask (or the current default mask).

```
RESTRICTmask <mask> (bool)
```

ANDs the specified Boolean expression to the value of the specified mask (or the value of the current default mask).

```
See: CONCEPT PARAMETRIC FILES
     COMMAND PUSHMASK              (Controls default mask name)
```

MASKNAME see TAGNAME

MASKS

```
MASKS
```

Prints (stacks) the names of the current masks.

```
See: MACRO   SHOWMASK              (Generates file showing mask names and values)
     CONCEPT PARAMETRIC FILES
```

MAXLENGTH see LENGTH

MAXNEXT

```
<-> MAXNext
```

Prints (stacks) the maximum useful count that could be used in a NEXT command (any larger count would be an attempt to go beyond the dummy EOF line). This is the number of lines in the logical file between the current line and the dummy EOF line, including the current line whether or not it is excluded from the logical file.

The result of -MAXNEXT is the line number of the current line if its status isn't DUMMY and no masks are set. However it should be remembered that the LINE command refers to the physical file, while MAXNEXT refers to the logical file.

Primarily for use by macros.

```
See: COMMAND LINES                (Related command)
     CONCEPT PARAMETRIC FILES
```

MERGE

```
<-> MERGE <tag>
          n
          *
```

Any totally identical versions within the specified range are merged. The default range is 2.

See: CONCEPT PARAMETRIC FILES

MODE see NAME

MOVEFROM

```
<-> MOVEFrom tag1 tag2
<-> COPYFrom  n1   n2
              *    *
```

Moves (or copies) the specified range after the current line. The last such line becomes the current line. The range is specified by identifying two lines, the first and the last, as follows:

Tag:	The line that has been associated with that tag.
Number:	The line that is that far forward from the current line.
*:	The dummy EOF line.

If the lines are moved, they keep their identity (in the sense of tags and the UNGO command). Lines can be copied, but not moved, from another file.

```
See: CONCEPT TAGS
     MACRO   MOVE            (A more flexible facility for moving and copying)
     COMMAND DESERIALIZE     (Controls serialization field to be cleared)
```

MSG see SAY

NAME

```
NAMe <fn <ft <fm>>>
FN
```

Set the file name (and type and mode, if specified) for writing file on disk.

```
TYPe <ft>
FT
```

Set the file type for writing file on disk.

```
MODe <fm>
FM
```

Set the file mode for writing file on disk. If mode letter only is given, the existing mode number is retained.

All of these will print (stack) the value for the current file is no argument is specified. If '*' is specified, it is interpreted literally (a dummy file) even though it is not a legal CMS file name.

NEXT

```
<-> Next <n>
    DOWN  *
```

Redefines the current line to be a line later in the file by n lines. If n is absent, it is assumed 1 (advance to the next line). If '*' is specified or n is too big, the dummy, EOF, line past the last line of the file becomes the current line and a return code of 1 is provided macros. If masks are set, only lines in the logical file are counted, except the current line counts no matter what its status.

```
<-> Up       <n>
    PREVious  *
```

Goes to a line in the opposite direction as NEXT: to an earlier line in the file (-UP is equivalent to NEXT).

```
See: CONCEPT CURRENT LINE
     CONCEPT DIRECTION
     COMMAND SCROLL             (Similar with specification by displayed lines)
```

NORESTORE

```
NORESTore
```

Only makes sense issued by a macro. Prevents the restoration of the file values saved prior to the macro call.

See: CONCEPT MACRO

NULL see BLANK

OVERLAY

Overlay <line>

Line specified: Modifies the character positions in the current line that correspond to the non-blank characters in the specified line to be those characters. An '_' in the specification forces a blank into the corresponding position of the current line.

No line specified: Reads the line separately and is otherwise the same.

SUperimpose <line>

The same as OVERLAY except '_' is not treated specially.

```
See: COMMAND CASE            (Controls the handling of lower-case letters)
     COMMAND DESERIALIZE     (Controls serialization field to be cleared)
     COMMAND IMAGE           (Controls the handling of back-space and tab)
     COMMAND TRUNC           (Controls the part of the line involved).
```

PFKEY

PFkey n <text>

Text specified: Sets the specified PF-key to the text, which should normally start with 'IMM ' and end with a blank.

No text specified: Prints (stacks) the current value of the PF-key.

Each file may have its own PF-key values.

Only the PF-keys 1 to 12 are supported.

TABKey n

Defines the specified PF-key to be used to input the tab character.

```
See: COMMAND PFOFF              (Additional PF-key commands)
     CONCEPT INITIALIZATION     (PF-keys are often set by P-SETUP files)
```

PFOFF

PFOff n

Restores the specified PF-key to the value it had when when the current file began being edited and removes it from the PF-keys defined for that file.

PFREnew

All PF-keys under control of P-EDIT are restored.

```
See: COMMAND PFKEY      (Additional PF-key commands)
```

PFRENEW see PFOFF

POPMASK see PUSHMASK

PREVIOUS see NEXT

PRINT

```
<-> Print <n <p <q>>>
          *  *  *
```

Prints the lines in the specified range or just the current line. The last line printed becomes the new current line. The lines as printed will span column p through column q of the lines in the file.

The first line will be printed ... ...tter what its status. Thereafter only lines with status FIXED, UNFIXED or HIDDEN are printed or contribute to the count. Lines with a status other than FIXED are printed preceded with their status (UNFIXED...line).

See: COMMAND WINDOW	(Controls how much to print if not specified)
COMMAND STACK	(Similar command for macros)
CONCEPT STATUS	

Note -- This differs from the PRINT command in EDITOR in which the first column (p) could not be specified.

PRY

PRY

Used for debugging P-EDIT. Goes to PRY, which must be loaded. RESUME will return to P-EDIT.

PUSHMASK

PUSHMask

Adds one to the default mask name. The default mask name is an integer which is used as the mask name if none is specified.

POPMask

Resets the default mask and subtracts one from from the default mask name.

See: COMMAND MASK	(Used to set default mask)
CONCEPT PARAMETRIC FILES	

READ

READ <*> <interp1 <interp2 ... >>

Asterisk specified: Enters a new level of P-EDIT and reads commands until an EXIT command. Entering a new level adds one to the level count and saves down the synonyms defined by SYNONYM and the setting of SCOPE for later restoration by EXIT. The commands read will be interpreted as specified; if none are specified the default interpretation is used (Usually SYNONYM COMMAND MACRO).

No asterisk is specified: One command is read and interpreted as specified or by the normal interpretation. The return code of the READ will be the return code of the command read. Both the command read and the READ command are recorded in the list of commands remembered, in that order. If the command read runs out of storage or detects a user interrupt, the return code of -9 or -10 from the READ does not imply that the READ was not put on the list of remembered commands (this is the only exception).

See: CONCEPT RESOLUTION	(Meaning of interpretations)
MACRO RECEDIT	(Edits a new file at a new level)

description of EXIT and LEVEL

EXIT <retcode>

Terminates the current level of P-EDIT with the return code (or 0) as specified as a non-negative integer.

See: MACRO QUIT   (Automatically does EXIT when applied to file RECEDITed)

LEVEL

Prints (stacks) the current level of P-EDIT. This number appears in the header of the screen if not 1. The level is 0 during the execution of SETUP P-EXEC.

See: CONCEPT SCREEN FORMAT

description of CALLER

CALLER

Returns as a return code the current mode of command execution:

```
     0 - Normal (READ *)
     1 - READ (with no asterisk; usually treated the same as 0)
     2 - Subcommand (from a macro or other EXEC2 file)
```

Primarily for use by macros.

RECFORM see FORMAT

RECLAIM

RECLAIM

Frees available storage and compacts storage in use. Prints (stacks) the message: n bytes ==> m bytes. Takes roughly 10 seconds.

FREE

Used to find out how much storage is still available without garbage collection. Prints (stacks) the message: n bytes left.

See: CONCEPT RECLAMATION

REMEMBER

```
REMEMber <n>
          *
```

Argument specified: Sets the number of commands to be remembered for UNGO and the like. If '*' is specified, no limit is imposed. REMEMBER * is assumed at the beginning of macros.

No argument specified: Prints (stacks) a line with two values: the current value and the default value (9 or whatever value was established by SETUP P-EXEC).

```
See: CONCEPT MEMORY             (How P-EDIT remembers commands)
     CONCEPT INITIALIZATION     (Default count can be set be SETUP P-EXEC)
```

REPLACE see INSERT

RESOLVE

RESolve command

Prints (stacks) the result of resolving the specified command in the current environment. If it is valid, the result is one of: a built-in command prefixed by the word 'Command', a macro file name, type and mode prefixed by the word 'Macro' (or 'Rejectmacro'), or a synonymous command prefixed by the word 'Synonym'. If not a valid command, 'Not found' is reported.

RESOLVEToken name <interp1 <interp2 ... >>

Similar to RESOLVE except the resolution order may be specified.

See: CONCEPT RESOLUTION

RESOLVEORDER

RESOLVEOrder <interp1 <interp2 ... >>
RESOLVORder

Argument specified: Resets the resolution order as specified. If the current editing level is 0 (SETUP P-EXEC is being executed), the default resolution order for user issued commands is also reset for the editing session.

No argument specified: Prints (stacks) the current resolution order.

See: CONCEPT RESOLUTION
     CONCEPT INITIALIZATION

RESOLVETOKEN see RESOLVE

RESOLVORDER see RESOLVEORDER

RESTRICTMASK see MASK

RESUME

RESUme <fnumber>

File number specified: Makes that file the current file. File numbers are the number displayed along with the file name in the WHAT field of the header.

No argument: Makes the previous file in the ring the current file. If there is no previous file, the last file in the ring becomes the current file.

RETRIEVE

```
RETRieve level n
```

Prints (stacks) the n-th command remembered from the specified level back. This is similar to the ? command except that commands at higher levels can be seen. A new level of memory of commands is entered whenever a macro is called or a READ * is issued. When a level exits, all commands remembered during it are merged into the memory of its effects. Thus, in a macro, RETRIEVE 1 1, will stack the command prior to the macro call. Primarily for use by macros.

```
See: CONCEPT MEMORY       (How P-EDIT remembers commands)
     COMMAND ?            (More convenient command for use by user)
```

REVERSE

```
<->     -     command
   REVerse
```

Many P-EDIT commands operate on the file in a direction (forward or backward). If these commands are prefixed with '-', they will work in the opposite direction; thus, -NEXT is the same as UP.

The REVERSE command is provided for macros because EXEC2 takes a line beginning with '-' as a label.

```
DIRECtion
```

Used by macros to test whether they were called with a '-' prefix. Returns a return code of 0 normally and a return code of 1 if it was called with a '-' prefix.

SAVE

```
SAVe  <fn <ft <fm <format> <width>>>>
        *   *   *
```

Saves the file on disk. Uses the given file designation, or the current file designation if none is given. Replace any file having the same file name and file type and the same mode letter. The current file designation is not changed.

The file is written on the disk (specified by fm) in the specified or current format and width. Each line written is padded with trailing blanks to the width if the format is F or FP. If the format is VB or V, the lines are written with one or no trailing blanks. V, VB, VP, VBP, and parametric FP format files appear as V format on disk, and the maximum line length of the file on disk may be less than the width.

error handling

For safety, the file is first written as a temporary workfile named 'P-EDIT CMSUT1 filemode'. Only when it is successfully written is the designated file erased and the workfile renamed to be 'filename filetype filemode'. This guards against loss of data in case the computer system fails in some way. No error during the operation can result in losing the old copy on disk without having written the new one.

```
See: COMMAND APPEND      (Appends to existing file)
     COMMAND FORMAT      (Defines format of file)
     COMMAND WIDTH       (Defines width of file)
```

SAVED

```
SAVED <YES>
      NO
```

P-EDIT remembers whether a file has been modified since it was last saved on disk. This is so that the macro QUIT can refuse to operate if called with a synonym when the current file has not been saved. Saving a file on disk, as the same file name as it is known to P-EDIT, will set SAVED to YES. Any modification to the file will set it to NO. Also, changing the file name, type or mode will set it to NO. If an argument is specified on the SAVED command, the SAVED flag will be set to it; if not, the current value of SAVED will be printed (stacked).

```
See: MACRO QUIT     (Stops editing current file permitting UNDO if not saved)
     MACRO OUT      (Stops editing all saved files and returns to CMS all)
```

SAVEVIEW

```
SAVEView fn <ft <fm <format <width>>>>
```

Saves the current view as the file, fn ft fm. Arguments that are unspecified or specified as '*', are taken to be the same as the current file.

SAY

```
SAY <text>
MSG
```

Text specified: Prints the text. If SCOPE is ON, it will appear in the message area when the screen is next refreshed.

No text specified: Reads the text separately and is otherwise the same.

```
SHOUT <text>
WARN
```

The same as SAY except the text is displayed bright on a 3270 and the alarm is sounded if permitted by ALARM.

These commands are primarily useful for macros.

```
See: CONCEPT SCREEN FORMAT
     COMMAND SCOPE           (Used to force messages to be displayed)
     COMMAND ALARM           (Controls ALARM mode)
```

SCALEX

```
SCALEX <width>
        *
```

Stacks a scale line of the specified width (no special marks for the ZONE and such). If the width is not specified, it will be taken as 80. If it is specified as '*', it will be taken as the WIDTH of the current file.

Primarily for use by macros.

```
See: CONCEPT SCREEN FORMAT
     COMMAND WIDTH           (Controls the WIDTH of the file)
     MACRO   SCALE           (Inserts a scale line into the file)
```

SCOPE

```
SCOpe    <ON>
          OFF
          NORM
          APL
          TEXT
         REFresh
   CHARS <continuation default>
```

No argument specified: Prints (stacks) a line describing how P-EDIT is currently using the display device (a 3270) consisting of two words: ON or OFF and NORM, APL or TEXT.

ON or OFF specified: Turns the use of the display device on or off. When the display is off, the P-EDIT operates as though a typewriter were being used, even if the device is a display device. If a real typewriter is being used, the display must be set off.

NORM, APL or TEXT specified: Resets the translation table to assume the appropriate character set is available on the display device. Unpredictable display behavior may result if the wrong translation table is selected; the NORM character set is a subset of the other two and thus is safe for all three displays.

CHARS specified by itself: Prints (stacks) a line containing two chracters: the current continuation and default characters. The continuation character, initially a '.', is the one which appears in triplicate at the beginning of continuation lines. The default character, initially '_', is displayed for each character that cannot be dislpayed on the display device.

CHARS specified with two characters: These characters can be non-blank single characters or the hexadecimal encoding of characters. The first is made the continuation character and the second the default character. These characters must be valid displayable characters: blank is permitted, but must be represented in hexidecimal (40).

REFRESH is specified: The display is immediately updated (assuming it is ON). This is mainly useful in edit macros.

All the SCOPE settings apply to all files being edited except CHARS, which applies to each file separately.

See: CONCEPT SCREEN FORMAT
     EXEC SETUP   (Standard setting of SCOPE)

SCREEN

SCREEN

Prints (stacks) the 24 lines most recently displayed on the display device. The input area will be blank, including the state that CP puts into the lower-right corner. Each line is preceded with '+' if it was displayed bright and with a blank otherwise If no screen has been displayed yet, 'Not found' is reported.

See: CONCEPT SCREEN FORMAT

SCROLL

```
<-> Scroll <<+>n>
             -
```

Line count specified: Goes to the line being displayed the specified distance from line 10 on the display device (where the current line starts). This is often more convenient than the NEXT and UP commands because it automatically accounts for lines in the file that occupy more than one line on the display and will ignore any hidden lines in parametric files.

No argument specified: Goes to the line as far away from the current line as will still permit at least one line to be redisplayed. This can be used to browse through a file in a fashion that guarantees all lines will be seen.

If SCOPE is OFF, SCROLL will position the current line to where it would be if ON assuming no messages in the message area.

```
See: EXEC    NUMBER                (The standard one permits SCROLL to be elided)
     CONCEPT SCREEN FORMAT
     CONCEPT PARAMETRIC FILES
```

SHOUT see SAY

SIMPLIFY

```
SIMPLIFY boolean
```

Prints (stacks) the simplified or canonical form of the Boolean expression specified. Any functions are expanded. Errors will generate an error message and 'TRUE' will be reported.

SORTX

```
<-> SORTX tag column length <Ascending>
          n     *      *     Descending
          *
```

Sorts the lines in the specified range according to the specified field. If the column is specified as a '*' it will be taken as '1' and if the length is specified as an '*' it will be taken as the WIDTH of the current file. If the option, DESCENDING, is specified, the lines will be in the opposite order. At the end, the current line will be the first line of the sorted lines.

```
See: MACRO   SORT          (A more flexible facility for sorting)
     COMMAND DESERIALIZE   (Controls serialization field to be cleared)
```

STACK

```
<->STACK <Fifo> <n <p <q>>>
          Lifo    *  *  *
```

Stacks the lines in the specified range. The lines are stacked FIFO (default) or LIFO. The words FIFO and LIFO may appear before or after n, p and q. The last line stacked becomes the new current line. The number of lines (n) defaults to 1. The lines as stacked will span column p through column q of the lines in the file. The defaults for p and q (denoted by '*' if necessary) are 1 and the column set by the TRUNC command. Primarily for use by macros.

If the number of characters to be stacked exceeds 254, they are broken up into 255 long segments, terminated by a shorter segment or null line if necessary (unless there are 257, the largest string, of them). In LIFO mode, the segments are stacked in the reverse order so they will be in a logical order when read. If any line stacked gets this treatment, a return code of 4 is returned.

STATUS

```
<-> STATus <tag <indicators>>
            n
            *
```

If indicators, a list of line status indicators, is specified, finds the next occurrence of a line with that status, within the specified range. If not, prints (stacks) all status indicators for the lines in the range. Each indicator can be abbreviated: Fixed, Unfixed, Hidden, EXcluded, Dummy, Top, EOf.

See: CONCEPT STATUS

STORE see FETCH

SUPERIMPOSE see OVERLAY

SYNONYM

```
SYNonym <name synonym minimum>
         name minimum
```

Defines the specified synonym as a synonym for the specified name. It may be abbreviated down to a prefix as small as the number specified. Synonyms are looked for only when the resolution order specifies so; normally on for user and off for macros. The standard SETUP P-EXEC defines a large number of synonyms, many of them necessary for macros to act as documented.

If SYNONYM has no arguments specified, it will print (stack) a line for each synonym currently defined.

See: COMMAND	UNSYNONYM	(Removes definition of synonym)
MACRO	SHOWSYNS	(Edits a file showing current synonyms)
CONCEPT	RESOLUTION ORDER	(Control over when synonyms apply)
EXEC	SETUP	(Defines initial synonyms)

TABKEY see PFKEY

TABS

```
TAbs <tab1 <tab2 ... >>
```

SETS the given tab columns or prints (stacks) the current tabs if no argument is specified. Tab settings are effective only if IMAGE is ON.

If the console is a 3270, the PF10 key is set to give the effect of a tab key, and the tab settings are honored in the input area of the screen.

```
See: CONCEPT IMAGING
     COMMAND IMAGE
```

TAG

```
TAG name
```

Makes the name be a tag referring to the current line.

```
UNT␣g name
```

Makes the name no longer be a tag.

```
GO name
```

Makes the line tagged by the name be the current line (redefining the current file, if necessary). Reports 'Not found' if the tag is undefined.

```
See: CONCEPT TAGS
     COMMAND TAGS       (Print tags defined over specified range)
     MACRO   RANGE      (Set default tags)
     MACRO   FINDTAG    (Search for tag specified as a pattern)
```

TAGGED see TAGS

TAGNAME

```
TAGName  string
VARName
```

The first name is found in string. It is stacked followed by the remainder of the string. If string is blank, 'Not found' is reported. A name begins with a letter and is followed by alphanumerics and '.', with blanks ignored around a '.' and and consecutive '.'s not permitted.

```
MASKName string
```

Same as TAGNAME except a positive integer is also accepted.

TAGS

```
<-> TAGS <tag>
         n
         *
```

Print (stack) all tag names for the current line and the specified range, if specified. The definition of the current line is not changed. If no tags are found 'Not found' is reported.

```
<-> TAGGED <tag>
```

Report 'Not found' unless the current line is tagged with the specified tag or any tag if none is specified. Used primarily by macros.

See:	CONCEPT TAGS	(Print tags defined over specified range)
	COMMAND TAG	(Tag current line)
	MACRO RANGE	(Set default tag)
	MACRO FINDTAG	(Search for tag specified as a pattern)

TOKEN

```
TOKen string
```

The first token is found in specified string. It is stacked followed by the remainder of the string. If string is blank, 'Not found' is reported. A token is a sequence of consecutive digits or such a sequence of letters or a single special character. Primarily for use by macros.

```
DELIMSTRing string
```

Same as TOKEN except a delimited string if found. A delimited string is any character (often '/') bracketing any characters not containing it. The closing bracket may be omitted. Brackets are not stacked.

TOP

```
<-> Top
```

The dummy line before the entire file, TOP, becomes the current line. The command -TOP goes to the dummy line, EOF.

```
<-> Bottom
```

The line in the logical file just before the dummy line after the entire file, EOF, becomes the current line. The command -BOTTOM goes to the first line in the logical file.

TRANSINX

```
TRANSINX <SET>
         CLEAR
```

SET specified: Reads four lines of 128 characters each and sets the input translation table to the first two and the input area translation table to the second two. Whenever a line is read, it is translated (based on an indexed look-up by EBCDIC code) character by character according to the input translation table. Whenever a line is put into the input area, it is translated according to the input area translation table (normally the inverse).

CLEAR: Input translation is turned off.

No argument specified: The four lines corresponding to the input translation are printed (stacked) in the reverse order (so LIFO stacking works nicely).

```
See: MACRO   TRANSIN        (Normal way to change the input translation)
     COMMAND TRANSOUTX      (Sets the output translation table)
```

TRANSOUTX

```
TRANSOUTX <SET>
          CLEAR
```

SET specified: Reads two lines of 128 characters each and sets the output translation table to them. Whenever text is printed or displayed, it is translated (based on an indexed look-up by EBCDIC code) character by character according to the output translation table.

CLEAR: Output translation is turned off.

No argument specified: The two lines corresponding to the output translation are printed (stacked) in the reverse order (so LIFO stacking works nicely).

```
See: MACRO   TRANSOUT       (Normal way to change the output translation)
     COMMAND TRANSINX       (Sets input translation tables)
```

TRUNC

```
TRUnc <q>
       *
```

The truncation column of a file limits the part of a line that P-EDIT commands that deal with a line as a whole operate on (such as INSERT and OVERLAY). It also provides the default extent for the STACK command.

Arguments specified: Sets the truncation column to q of slaves it to the width if '*' is specified. If the truncation column is slaved to the width, it will be equal to the file width even if that later changes.

No argument specified: Prints or stacks the current truncation setting as two items: truncation column or '*' and truncation column or file width.

```
See: COMMAND WIDTH      (Maximum width of line written to disk)
     COMMAND ZONE       (Limit for commands that operate on text)
     COMMAND STACK      (Used by macros to stack lines in the file)
```

TYPE see NAME

UNDO

```
UNdo <n>
      *
```

This command undoes the effects of the previous n remembered commands. If issued in a macro, only commands issued by the macro can be undone. If no argument is specified, one command is undone; if '*' is specified all remembered commands (at the current level) are undone (used by macros). There is a limit of how many commands are remembered. If the specified command is not remembered, 'Not found' is reported. Any changes to the CMS file system or the CP virtual machine are not, of course, undone.

See:	CONCEPT MEMORY	(How P-EDIT remembers commands)
	MACRO BACK	(Similar command for additional undoing aftcr an UNDO)
	COMMAND UNGO	(Similar command that only undoes positioning effects)

UNGO

```
UNGo <n>
      *
```

This is similar to UNDO, except only those effects of the previous n commands in positioning the current line are undone. Thus UNGOing a positioning command (such as NEXT) is equivilent to UNDOing it. If issued in a macro, only positioning done by the macro can be undone. If no argument is specified, the positioning of the previous command is undone; if '*' is specified the positioning of all remembered commands (at the current level) are undone (used by macros). There is a limit of how many commands are remembered. If the specified command is not remembered, 'Not found' is reported. If the current line restored is in a different file, that file is resumed. If that file has been dropped, it is re-activated.

See:	CONCEPT MEMORY	(How P-EDIT remembers commands)
	COMMAND UNDO	(Similar command that undoes all effects of commands)

UNMASK see MASK

UNSTORE see FETCH

UNSYNONYM

```
UNSYNonym synonym
```

The specified synonym is removed.

See:	COMMAND SYNONYM	(Defines synonyms)

UNTAG see TAG

UP see NEXT

VARNAME see TAGNAME

VARS

```
VARS
```

Prints (stacks) the names of all defined variables.

All the above are primarily for use by macros.

```
See: CONCEPT VARIABLES
     MACRO   SHOWVARS       (Generates a file showing defined variables)
```

VERIFY

```
VERify <ON>
       OFF
```

Sets verify mode or prints (stacks) the current setting if no argument is specified. Verify mode is intended for when using a typewriter as a display device. If the display device is a 3270, there is no reason to have it on. In verify mode, P-EDIT prints each line that is located or changed. In addition, whenever the left most field of the screen's header line (such as Edit:) would have been changed, the new value is printed. If the screen header would have been displayed bright (to warn that the masks are not at their base value) the header will be preceded with 'MASKS...'.

The part of the line which is printed is the same part which is specified in the WINDOW command. Thus, when the display device is ON, the columns of the lines verified are the same columns as shown on the screen for the body of the file. If the display device is OFF, the WINDOW command may be used in conjunction with the VERIFY command to select the columns to be verified.

VERSION

```
VERSion
```

Prints (stacks) the message: P-EDIT version as of time, date. This is the time of the compilation of this version of P-EDIT.

VERSIONS

```
<-> VERSIONS <tag>
              n
              *
```

Computes the minimum Boolean expressions necessary to generate each version of the lines in the specified range, given the current masks. These are then printed (stacked). If there are no unfixed lines in the range, the single Boolean expression, TRUE, is reported. Used primarily by macros.

```
See: CONCEPT PARAMETRIC FILE
     MACRO   SHOW              (Steps a mask through each version)
```

VIEW

```
VIEW <boolean>
```

Restrict or print (stack) the current view. If a Boolean expression is specified, it becomes the most preferred for purposes of determining the view seen on the screen.

The current view is defined by a Boolean expression (the one printed or stacked) and a priority list of Boolean expressions. When the status of a line is needed by P-EDIT, it can happen that the current view isn't sufficient to resolve it (both it and its NOT are consistent with the view). When this happens, the priority list is searched (in order) to see whether it contains a Boolean expression which would help if ANDed to the view. If so, the view is restricted and the process continues until the status of the line is resolved.

If this process isn't sufficient, P-EDIT must make an arbitrary choice whether to place the line in the view or not. It always chooses to do so: it restricts the view with the Boolean expression associated with the line and adds it to the end of the priority list (low priority) so that it will tend to make the same choice next time.

When the user specifies a preferred view, using the VIEW command, the specified Boolean expression is added to the beginning of the priority list (high priority). Whenever he changes the view or a mask, the view is temporarily set to only the AND of the current masks. Thus the next attempt to find the status of an unfixed line will cause the recomputation of the view. Note: the following command will print (stack) the AND of the current masks:
GROUP .VIEW TRUE.VIEW.

See: CONCEPT STATUS

WARN see SAY

WIDTH

```
WIDth <n>
      *
```

Sets the width of the file being edited; or, if issued with no argument, print (stack) the current width. If '*' is specified, the width is set to the maximum possible.

The width of the file in P-EDIT governs the width of the file when written on disk (such as SAVE) or read from disk (such as GETFILE). It also determines the actual truncation column when 'TRUNC *' is in effect, and the actual end zone when 'ZONE p *' is in effect. Reducing the width below the current maximum line length will provoke a warning message as a file may not be saved in that state.

The width of a file is usually not changed during editing, as the correct width for most situations is set when editing the file is begun.

WINDOW

```
Window <p <q>>
          *
```

Set one or both of the columns between which (inclusively) the lines of the file are displayed on the screen; or, print (stack) the current settings if no argument is given. Columns included in the window but not in the line are displayed as blanks, but trailing, all blank continuation lines are not displayed.

If the second column is not specified it will remain the same. If it is an '*', it will become the maximum value (65535). If the WINDOW becomes wider than 850 characters, only the first 850 will be displayed; this allows the current line to occupy 11 lines on the screen, leaving room on the screen for the first part of the line following the current line.

Particularly useful for looking at LISTING and other wide files.

X

```
X <command>
Y n
  *
  ?
```

Command specified: Associates the command with the specified abbreviation (X, Y or whatever synonym was used). This is done by setting the P-EDIT variable, COMMAND.X (or COMMAND.Y or COMMAND.synonym), to the command.

Number specified: Executes the command previously associated the specified number of times or until an error (non-zero return code) results.

No argument specified: Executes the command previously associated once.

'*' specified: Executes the command previously associated until an error (non-zero return code) results.

'?' specified: Prints (stacks) a line with two fields, the name of the abbreviation and its associated value. If SCOPE is ON, this is placed into the input area where local editing can modifiy it and pushing ENTER will redefine that abbreviation.

See: MACRO SHORT	(Defines some other short name for a command)
CONCEPT VARIABLES	(Description of P-EDIT variables)
COMMAND SCOPE	(Controls meaning of '?')

XFERCONTROL

```
XFERControl fnumber
```

The control of the file whose file number is specified, becomes shared with the current file.

The data structure in P-EDIT that contains the masks, their values, and the view information of a parametric file is called the "control". Initially each file gets its own control. Often it is desirable for more than one file to share a control; say THEY are part of the same application.

When a control is transferred, it is "merged" with the current control. This merger consists of ANDing together masks with the same names, except masks with numeric names (default masks) are relocated so that the specified control dominates (has higher names). The view information is also merged so that the SPECIFIED control dominates.

See: CONCEPT MASKS
     CONCEPT VIEW

XMACRO see MACRO

Y see X

ZONE

```
Zone <p <q>>
      *  *
```

The zone of a file consists of two columns which limit (inclusively) commands which do contextual searches (such as LOCATE). Those commands which also modify the file (such as CHANGE) are also restricted to the zone in their modification and will terminate with the message "Truncated" (return code of 4 for macros) if necessary.

Arguments specified: Sets the beginning of the zone to column p or 1 if '*' is specified; sets the ending of the zone, if specified, to q or slaves it to the width if '*' is specified. If the ending of the zone is slaved to the width, it will be equal to the file width even if that later changes.

No argument specified: Prints or stacks the current zone setting as three items: beginning column, ending column or '*', and ending column or file width.

See:	COMMAND WIDTH	(Maximum width of line written to disk)
	COMMAND TRUNC	(Limit for commands that operate on lines)

< see RIGHT

$

```
$ <command>
```

Command specified: Executes the CP or CMS command. If the command prints any output it is likely to flash by faster than it can be read.

No command specified: Reads the command separately and is otherwise the same. Thus, the if a CP or CMS command is issued by mistake without a '$' prefix, entering '"$' will execute it properly.

```
See: COMMAND "      (Stacks previous command)
     MACRO $        (Similar command that protects against lost output)
     MACRO CP       (Similar command for CP that uses message area for output)
```

_

```
_ <command>
```

Command specified: Executes the CP or CMS command after clearing the screen and printing it. Following its execution, 'R;' is printed (if it had a non-zero return code, that precedes the ';' in parentheses). CANCEL must be pushed to return the screen to P-EDIT.

No command specified: Reads the command separately and is otherwise the same. Thus, the if a CP or CMS command is issued by mistake without a '_' prefix, entering '"_' will execute it properly.

```
See: COMMAND "      (Stacks previous command)
     MACRO $        (Similar command that doesn't need CANCEL)
     MACRO CP       (Similar command for CP that uses message area for output)
```

> see RIGHT

' see INDENT

= see EVAL

ADD

```
ADd text
```

Adds the specified text after the last non-blank character on the line.

Note -- An extra blank (total of two) is needed before the text to add a a word to the line.

```
See: COMMAND CASE       (Controls handling of lower-case letters)
     COMMAND IMAGE      (Controls handling of back-space and tab)
     COMMAND TRUNC      (Controls part of line considered)
```

The standard SETUP P-EXEC defines the necessary synonym.

ADJUST

```
<-> ADJUST <<+>delta <n <m>>>
            -delta   *
```

Adjusts the indentation of the lines in the specified range so that they line up with each other and are offset by a delta, as specified, from the line before them. A positive delta is specified for more indentation and a negative one for less.

ADJUST normally operates only on the field in the lines from column 1 to the truncation column. However, if a third argument is specified, it will be taken to be the starting column of that field.

```
See: COMMAND TRUNC      (Controls the last column of the field)
     MACRO   INDENT     (Useful for entering the lines correctly originally)
```

AGAIN

```
<-> AGAIN <n>
```

Re-executes the n-th remembered command back (1, the default, is the previous command).

Note -- If the re-executed command itself needs the list of remembered commands, 'Not found' will be reported.

```
See: CONCEPT MEMORY     (How commands are remembered)
     COMMAND "          (Similar command that stacks the previous command)
```

ANNEX

```
<-> ANNEX line
```

Inserts the specified line (or blank, if none) after the current line only for those versions of the file that contain the current line. This new line becomes the current line. Only different than INSERT for parametric files.

```
See: CONCEPT PARAMETRIC FILES
     COMMAND CASE                (Controls handling of lower-case letters)
     COMMAND IMAGE               (Controls handling of back-space and tab)
     COMMAND TRUNC               (Controls how long a line can be inserted)
```

The standard SETUP P-EXEC defines the necessary synonym.

ARCV

```
ARCV    <n>
     arguments
```

No argument or number specified: Attempts to archive or retrieve the files designated by the lines in the range. If the current file type is LIST the files are archived, if ARCHIVE, they are retrieved. The lines are modified appropriately.

ARCV arguments specified: Invokes ARCV EXEC with those arguments.

```
See: MACRO FILELIST        (Generates LIST files of CMS files)
     MACRO SORT            (Sorts LIST and ARCHIVE files)
```

ARGS

```
<-> ARGS command
```

Executes the specified P-EDIT command after replacing any arguments that it contains with words picked up from the current line (often in LIST file format). Permitted arguments are &n, &FN, &FT, &FM, &NAME, &TYPE and &MODE.

If one of the file references are used, the command is not executed if the current line is a comment line or the values picked up are not syntactically correct; no error is reported. Argument names may be upper or lower case, but not mixed. Unlike EXEC2, blanks are preserved in the command and must separate arguments.

For example, the following will copy the files named in the next 5 lines to the A-disk and update the LIST file:

```
     FOR 5 ARGS ;$QCOPY &FN &FT &FM A;CHANGE / &FM / A;CURRENT
```

```
See: MACRO FILELIST       (Generates LIST files)
```

ASMCOM

```
ASMCOM <n>
        *
```

Moves comments in assembly code so that they start in column 31 or two spaces beyond the last operand.

AVAND

```
<-> AVAND /string1</string2 ... ></>
```

Goes to the next occurence of a line after the current line missing one of the specified strings. The delimiter (shown here as a '/') may be any non-blank character not in any of the strings. AVAND will run somewhat faster if the least likely strings are specified early. Two strings in a line can overlap; thus AVAND /ABC/BCD/ will skip the line containing the string 'ABCD'.

```
See: COMMAND CASE      (Controls handling of lower-case letters)
     COMMAND ZONE      (Controls the part of the lines searched)
     MACRO   LAND      (Locate next line with all specified strings)
     MACRO   AVOR      (Locate next line missing all specified strings)
     MACRO   LOR       (Locate next line with any specified string)
```

AVOR

```
<-> AVOR /string1</string2 ... > </>
```

Goes to the next occurence of a line after the current line containing none of the specified strings. The delimiter (shown here as a '/') may be any non-blank character not in any of the strings. AVOR will run somewhat faster if the least likely strings are specified early.

```
See: COMMAND CASE      (Controls handling of lower-case letters)
     COMMAND ZONE      (Controls the part of the lines searched)
     MACRO   LOR       (Locate next line with any specified string)
     MACRO   AVAND     (Locate next line missing any specified strings)
     MACRO   LAND      (Locate next line with all specified strings)
```

BACK

```
BACK <<+>n>
       -
```

Undoes the specified number of P-EDIT commands more (or less) than have already been undone. BACK can be confused by coverted BACK and UNDO commands.

Thus, the following pairs of commands have very different meanings:

```
                   UNDO        (Undoes nothing)
                   UNDO

                   BACK        (Undoes previous two commands)
                   BACK
```

BACK is a good candidate for a PF-key.

```
See: CONCEPT MEMORY
     COMMAND UNDO       (Undoes an absolute number of commands)
```

BALANCE

```
<-> BALance <n>
             *
```

Checks the specified expression for balanced parentheses (or n expressions or all expressions). An expression is all lines from the current line to the line before the first line of the next expression. The next expression is deemed to begin with the next line that has a left parenthesis before or at the column of the first left parenthesis of the current expression.

```
<-> PArentheses <tag>
                 n
                 *
```

Checks the lines in the specified range for balenced parentheses.

A line that is filled with right parentheses is treated like one right parenthesis. The standard SETUP P-EXEC defines the necessary synonyms.

```
See: COMMAND ZONE     (Controls the part of each line examined)
```

BLIND

```
<-> BLIND
```

Enters blind input mode. This is just like input mode except the screen is not updated automatically. A PF-key can be used to cause the screen to be updated. Otherwise, the screen can be updated by leaving blind input mode and re-entering it ('#BLIND').

The standard SETUP P-EXEC defines the PF-key to use.

BOOL

```
<-> BOOL   <n>
            *
        command
```

Range specified: Prints the Boolean expressions asso~iated with the lines in that range. The last line becomes the current line.

Command specified: Proforms that P-EDIT command upon the Boolean expressions in the current file. This is essentially the same as setting EDITMODE to BOOLEAN and issuing the command, except all sorts of file values (ZONE, CASE and such) are set properly to act on Boolean expressions.

```
See: CONCEPT PARAMETRIC FILES
     COMMAND EDITMODE
```

BURN

```
BURN
```

Erases the file you are in and QUITs.

CANCEL

```
CANCEL
```

Cancel the entire edit session and return to whoever invoked P-EDIT.

Warning: no files in the ring are saved.

CHINDENT

```
<-> CHINdent /string1</string2</ <tag <zone1 <zone2>>>>>
                                  n      *       *
                                  *
```

Changes the characters used to indent the lines over the specified range. Each line is examined to determine the number of occurences of the first specified string at the beginning of the line. These are changed to the same number of the second specified string. The part of each line examined can be specified; if not, the current zone settings are used. Examples:

```
CHINDENT /  /. / *           Changes the indentation to . . . . . . . .
CHINDENT /  /. / * 2         Same except the first column is ignored
CHINDENT / /  / *            Doubles amount of indentation
CHINDENT / // *              Unindents all lines
```

CMCASE see MCASE

COMPARE

```
COMPARE <fn1 ft1 fm1 fn2 ft2 <fm2>> <(<option1 <option2 ... >><)>>
        fn1 ft1 fn2 <ft2>
               fm2

Options: Margins ml mr <ml2 mr2>
         Narrow
         Wide
         Width w
         Cc
         Quiet
```

Compares the two specified files with each other, producing a result file which lists the differences between the two files in parallel columns. If the result file contains any differences it is edited, otherwise a message is displayed. (Nothing is written on disk by this macro.)

File name specified: Any missing file name, type or mode defaults to the corresponding one specified (mode of A is assumed if none specified). Thus: COMPARE A B C, compares file A B A and file C B A.

Simply mode specifed: The first file is taken from the current line (in the format produced by FILELIST). The second file has the same name and type, but has the mode specified.

No file specification: The first file is taken from the current line and the second file is taken from the next line.

description of options

Normally, a message is displayed which says that no differences were found, or how many (blocks of) differences were found. The QUIET option suppresses this message.

The MARGINS between which the two files are compared may be specified as an option (ml1 is left margin of file 1, ... mr2 is right margin of file 2). Default margins are 1 through * for both files (except 1 through 72 for ASSEMBLE type files). The WIDTH of the data portion of the output columns may be specified (between 30 and 119). If the CC option is present, the result file contains carriage control characters in column 1 so it is suitable for printing. The default NARROW option implies 'WIDTH 32', the WIDE option implies 'CC WIDTH 58'. The total width of NARROW files is 80, of WIDE files 133 (CC included).

The example that follows shows a typical output file in the NARROW format. Lines in the output file which contain only '*' delimit the header (which appears on each page of a CC result). Lines which contain only '=' indicate that both source files contain equal-within-the-margins lines in that range. A field prefixed by 'n.' displays (the left part of) the n-th line in the corresponding source file. A field prefixed by 'n/' displays the n-th line and indicates that it appears AFTER the lines listed in the opposite column. The data portion of a field starts after one space past the 'n.' or 'n/' and continues through the character position marked by the last '*' in the header (or '=') over that field. A blank field indicates that this file has no lines.

example

```
E-DISPLY LISP B1  80/02/22 04:49         E-DISPLY LISP2 B1  80/01/19 11:57
  V 81  553 lines, margins 1-80            V 81  553 lines, margins 1-80
***************************************  ***************************************
=======================================  =======================================
                                           192.      (ITER (INIT I 80)
                                           193.            REPEAT
  192/      (COND                          194.         (SETQ I ($-1 I)) )
=======================================  =======================================
  204.               (CHR "%") )           211.                (CHR "#") )
  205.            (ELSE (CHR "/")) )       212.             (ELSE (CHR "|")) )
  206.      (COND ( (PLTP EG-WINDOWL 80    213/      (SETQ ED-DISPLAYFLAGS (LOGO
=======================================  =======================================
  300.         (COND ( (PGTP S ED-WINDOW   305.         (COND ( (PGTP S EG-WINDOW
```

This macro uses the executable module QKOMPARE 370 (after loading it into CMS free storage) and thus runs MUCH faster than the old KOMPARE macro when a significant number of differences are present. The maximum width permitted an input file is 65535.

See: MACRO MORE	(Shows more detail of the two files)
MACRO FILELIST	(Generates LIST type files)

CONFLICT

```
<->CONFLICT
```

Finds the next occurence of an unfixed group of lines which contains lines of both UNFIXED and HIDDEN status. This is an indication of conflict between the current view and other possibilities.

CONSOLE

```
CONSOLE
```

Edits the console spool file since last used. Unlike _CONSOLE LOOK, nothing is written onto disk: the console spool file is held in the virtual card reader.

For information on console spooling, enter: _CONSOLE ?.

COPY see MOVE

CP

```
CP <command>
```

Command specified: Executes the CP command. Any output will be routed to the message area as is the command and an indication of the return code that it returned.

No command specified: Reads the command separately and is otherwise the same. Thus, the if a CP command is issued by mistake without a 'CP' prefix, entering '" CP' will execute it properly (the initial blank is needed to prevent CMS from interpreting the line as the immediate command, 'CP').

See: COMMAND "	(Stacks previous command)
MACRO $	(Similar command for CMS commands which do not print)
MACRO _	(Similar command for CMS commands that might print)

CPLIST

```
CPLIST  <*>
        userid
```

User ID is missing, '*' or same as current user: Creates and edits a file showing information about the reader, printer and punch spool files. The lines are in the same order of that given by the CP command, QUERY, so the newest files, within device type, will be last assuming the ORDER attribute has not been altered.

User ID is another user: Creates and edits a file showing information about all reader files for that user that originated with the current user. The order of those files in the other user's reader relative to other files will be changed.

See:	MACRO P-EDIT	(Edits the file designated by the current line)
	MACRO ERASE	(Purges the files designated by the specified lines)
	MACRO ORDER	(Re-orders the files designated by the specified lines)
	MACRO RIGHT	(Used to see DIST, TBL and TAG information)

warning

If another user transfers a file you sent him back to you during the operation, it will be treated as though it came from the specified user. This condition can be detected but not prevented. A warning will be printed if it occurs. This is a very unlikely occurrence and should it happen, CP would have told you about the transfer.

CUF

```
CUF /string1</string2</>>
```

Locate the unique occurrence of the first specified string in the current file and change it to the second. 'Not found' or 'Not unique' is reported if so (return code of '1' for macros). This operation never repositions the current line.

The delimiter, shown here as '/', may be any non-blank character not in the string.

See:	COMMAND CASE	(Controls the handling of lower-case letters)
	COMMAND ZONE	(Controls the part of the lines considered)
	MACRO LUF	(Analogous LOCATE command)
	MACRO CUS	(Changes unique string on screen)

CUS

```
CUS /string1</string2</>>
```

Locate the unique occurrence of the first specified string on the screen and change it to the second. 'Not found' or 'Not unique' is reported if so (return code of '1' for macros). If SCOPE is OFF, 'Not found' is reported. This operation never repositions the current line.

The delimiter, shown here as '/', may be any non-blank character not in the string.

Characters not being displayed because they are lost due to lines being displayed bright, are still considered to be "on the screen".

See:	COMMAND CASE	(Controls the handling of lower-case letters)
	COMMAND SCOPE	(Controls whether SCOPE is ON)
	COMMAND WINDOW	(Controls the part of the lines considered)
	MACRO LUF	(Analogous LOCATE command)
	MACRO CUF	(Changes unique string in file)

CUT

```
<-> CUT /string</>
        column
```

Cuts the current line at the first occurrence of the specified string or column, making two lines of it. The second part of the old line becomes the new current line. The delimiter (shown here as '/') may be any non-blank, non-numberic not in the string.

```
See: MACRO PASTE      (Opposite of CUT)
     MACRO K          (Similar to CUT)
```

DISKS

```
DISKS
```

Creates and edits a file containing information about currently accessed disks. A number of commands operate on such files.

```
See: MACRO DROPDISK     (Drop the disk using DROP EXEC)
     MACRO LINKS        (Insert lines showing all links to the disk)
     MACRO REACCESS     (Reaccess the disk so the directory is current)
```

DOTHERS

```
<-> DOTHERS <command>
```

Deletes all lines not "satisfied" by the specified command. That command should be in the "LOCATE" family: namely those commands that search for something and give a return code of '1' if not found. Because this macro repeatedly executes the command starting at different lines, a range specified by a line count (rather than '*' or a tag) will not work. Examples:

```
DOTHERS /APPLE/            Deletes lines not containing 'APPLE'
DOTHERS ¬PEAR¬ X           Deletes lines containing 'PEAR' until tag 'X'
DOTHERS LAND /APPLE/PEAR/  Delete lines not containing both 'APPLE' and 'PEAR'
```

for use of DOTHERS with masks

The DOTHERS command is most often useful following the setting of a mask. Thus:

```
MASK A=1
TOP
DOTHERS /PEACH/
```

will show you just the lines containing 'PEACH'. Before modifying any of these lines, be sure to:

```
UNMASK
```

or you will only change the file under the temporary assumption 'A=1'. DOTHERS modifies the current view so you will still see the same lines after the UNMASK. You may wish to use the HILITE command to modify the method of high-lighting.

To remove the changes made to the file:

```
               TOP
               MAKE * A=1 FALSE
```

or use UNDO.

The "LOCATE" family:

```
    COMMAND LOCATE      (Locates string)
    COMMAND AVIOD       (Locates absence of string)
    COMMAND FIND        (Locates characters at fixed columns)
    MACRO   LAND        (Locates occurence of all of specified strings)
    MACRO   LOR         (Locates occurence of any of specified strings)
    MACRO   AVAND       (Locates absence of any of specified strings)
    MACRO   AVOR        (Locates absence of all of specified strings)
```

DROPDISK

```
DROPDISK
```

Drops the disk named on the current line in the format generated by DISKS.

See: MACRO DISKS   (Generates DISKS files)

DTOP

```
<-> DTOP
```

Deletes all lines from the previous to the top (or next to the end). Prints the number of lines deleted.

DUMMY

```
<->DUMMY <tag>
```

Generates a dummy line after (or before for -DUMMY) the current line. If a tag name is specified, it becomes the tag of the dummy line. This is princi-ply used to give a name to the position between two lines.

DUP

```
<-> DUP <n <tag>>
           m
           *
```

No argument specified: Inserts a copy of the current line.

One number specified: Inserts that number of copies of the current line.

Number and range specified: Inserts that number of copies of the lines in the specified range.

In any case, the last line of the last copy becomes the current line.

See: COMMAND DESERIALIZE   (Controls serialization field to be cleared)

EMBED

```
<-> IMBED /string1</string2</ <tag>>>
                                *
                                n
```

Modifies the lines in the specified range by placing the first specified string to the left of each and the second to the right. The two strings are placed before the first and after the last non-blank characters. The delimiter (shown here as '/') may be any non-blank character not in the strings and, except for the first, is optional at the end of the command.

Examples:

```
     BOOL EMBED /A=1&(/)          Restricts the line by A=1
     BOOL EMBED /A=1|(/)          Promotes the line by A=1
     EMBED /COVERT /               Prefixes command with COVERT
```

See: MACRO BOOL

ERASE

```
<-> ERASE <tag tag>
           n   n
           *   *
```

"Erases" the files designated by the lines in the specified range of a LIST or CP-LIST type file. "Erasing" a file uses ERASE, HIDE, PURGE, or TRANSFER & PURGE depending on whether the file is on a read/write disk, is on a read-only disk, is a spool file, or is RDR spool file in another user's reader. The argument, if any, must be specified twice for safety. The lines corresponding to the erased files are changed into comment lines with blanks converted to hyphens.

```
See: MACRO FILELIST      (Generates LIST files)
     MACRO CPLIST        (Generates CP-LIST files)
```

ERF see P-EDIT

EVAL

```
EVAL <H> expression
=
```

Evaluates the specified expression using APL rules but FORTRAN operators ('*' means multiply) and prints (stacks) the result. For example, 1+2*(2+3) will be evaluated to 11.

If the first token of the expression is 'H', the entire compution is done in hexidecimal rather than decimal, and the result is returned in hexidecimal. If individual operands have an 'H' as their first character, they will be interpreted as hexidecimal, but the computation will still be in decimal.

The standard SETUP P-EXEC defines the necessary synonym.

EXCHANGE

```
<-> EXCHange /string1/string2</ <tag <p>>>
                                     n
                                     *
```

Exchanges the first (p-th) occurence of the first specified string with the next occurence of the second specified string on each line of the specified range. If one of the strings is not found on a line, no change to it is made. The second string may not overlap the first if it is to be recognized. Changed lines are recanonicallized if IMAGE is CANON. The last line scanned becomes the new current line.

The delimiter (shown here as '/') may by any non-blank character not in the strings. If the two strings are not found on any line, 'Not found' is reported.

```
See: COMMAND CASE           (Controls the handling of lower-case letters)
     COMMAND DESERIALIZE    (Controls serialization field to be cleared)
     COMMAND IMAGE          (Controls the handling of back-space and tab)
     COMMAND ZONE           (Controls the part of the lines considered)
```

FIELDBR

```
FIELDBRackets <left right>
```

Brackets specified: Sets the left and right brackets for FILLIN and INFILLIN. These can be any string of non-blanks. The default values are '<' and '>'.

No argument specified: Prints (stacks) the current values of the left and right brackets for the current file.

```
See: MACRO FILLIN       (CHANGE for fill-in fields)
     MACRO INFILLIN     (INCHANGE for fill-in fields)
```

The standard SETUP P-EXEC defines the necessary synonym.

FIELDBRACKETS see FIELDBR

FILE

```
FILe <fn <ft <fm <format> <width>>>>
```

Saves the current file as specified and stops editing it. A message is printed that describes the resulting file.

```
See: COMMAND SAVE     (For meaning of arguments)
     MACRO   QUIT     (FILE is essentially the same as SAVE following by QUIT)
```

The standard SETUP P-EXEC defines the necessary synonym.

FILELIST

```
FILEList <fn <ft <fm>> <(options>>
LISTFile
LF
```

This invokes the FILELIST module (cf. FILELIST MEMO) to generate the specified list of files which is then edited as a new file (or appended to the current file -- see below). If no arguments are provided, the information about the current file is displayed in the message area. If LT or LM are used as synonyms of FILELIST, one or two asterisks, respectively, are assumed to preceed the arguments. As in FILELIST EXEC, any of fn, ft and fm may be parenthesized lists of patterns. If fm is omitted, all read/write disks are searched instead of just the A disk. The permissible options are: Append, Name, TYpe, Mode, Big, Small, Old, NEw, SInce, Unique, ONe, and RW. Note that DISK, FIFO, LIFO and TERM are excluded, and APPEND means append to the current file being edited, not append to an existing disk file.

```
LT ft <fm> <(options>
```

Same as LISTFILE * ft <fm> <(options>

```
LM fm <(options>
```

Same as LISTFILE * * fm <(options>

The standard SETUP P-EXEC defines the necessary synonyms.

macros which operate on LIST files

MACRO ARCV	(Archives file designated by current line)
MACRO COMPARE	(Compares files designated by current and next line)
MACRO ERASE	(Erases file designated by current line)
MACRO P-EDIT	(Edits file designated by current line)
MACRO QCOPY	(Copies file designated by current line)
MACRO QMOVE	(Moves file designated by current line)
MACRO RENAME	(Renames file designated by current line)
MACRO RENEW	(Renews line of LIST file with latest information)
MACRO SAME	(Inserts similar descriptions as that on current line)
MACRO SEARCH	(Executes P-EDIT command for every file designated in range)
MACRO SORT	(Sorts lines in LIST file by file name, etc.)

FILLIN

```
<->FILLIN </string/> <tag <p>>
                      n    *
                      *    G
```

This is like the CHANGE command for fill-in fields. It is often desired to have a file (usually a FORM file) serve as a form to be filled out to produce similar SCRIPT files easily. These files have fields in them that must be filled in with specific information each time, called fill-in fields. A fill-in field consists of a left bracket, an optional default value, and a right bracket. These brackets default to '<' and '>' if not specified. They may be specified with FIELDBRACKETS. Example:

```
        <He> said to his <wife>, "I am going to the store".
```

The first (or all or p'th) fill-in field is located on each line of the specified range of lines. If a string is specified, it is replaced by that string. If not, it is replaced by the default value coded in the fill-in field itself. If the resulting line is null, it is deleted.

The delimiter (shown here as '/') may be any special character not in the string except '*' and is optional if at the end of the command.

See:	RESOLVE FIELDBRACKETS	(Set or query fill-in field brackets)
	MACRO INFILLIN	(INCHANGE for fill-in fields)
	COMMAND CHANGE	(More detail of arguments that are like CHANGE)

FINDTAG

```
<-> FINDTAG <pattern>
```

Searches the current file for the first occurence of the specified tag or if none specified, the next occurence of any tag. The tag may be specified with '*'s meaning any (or no) characters in the tag name at that position.

FIRST see RANGE

FOR

```
<-> FOR tag <command>
        n
        *
```

Command does not go to a different line: Executes the specified command once for each line in the specified range with the current line set to that line.

Command does change current line: Executes the specified command until it is determined that it is the last time or EOF is reached. After each execution except the last, a NEXT is executed. The determination of the last execution is done prior to its execution by the current line being the last in the file or there being no more lines prior to the specified tag.

Any error will terminate the loop.

```
<-> FORD <-> n <command>
```

S me as FOR except the display is updated after each execution of the command.

The standard SETUP P-EXEC defines the necessary synonym.

FORD see FOR

FORGET

```
FORGET <reserve>
```

Clears all remembered commands except for the most recent commands specified to be reserved. If no number of commands are specified to be reserved, all memory is cleared. Resets the REMEMBER count to the default value.

GETFILES

```
GETFILES <fn <ft <fm >>> <(options>
```

Inserts the contents of all the specified files followinf the current line. The arguments can be of any form permitted by FILELIST.

GETSYNONYMS see GETSYNS

GETSYNS

```
<-> GETSYNS <disk>
             *
```

Inserts into the current file the SYNONYM commands expected by the macros on the specified disk. This is useful for writing your own SETUP or PROFILE P-EXEC. If no disk is specified, it will be taken to be the same disk as the GETSYNS macro itself (normally the P-EDIT system disk with the label EDITOR).

Macros specify synonyms by a line starting with '*SYNONYMS' within the first 5 lines and followed by the synonyms, each optionally followed by the length of the minimum abbreviation. If the first synonym is missing (the first word is a number) the name of the macro will be assumed.

See: CONCEPT INITIALIZATION
     COMMAND SYNONYM

GETTAIL

```
GETTAIL <fn <ft <fm <m <n> <(Quiet<)>>>>>>
         *   *   *  *  *
```

Like GETFILE, except the line number, m, is taken to be the relative line from the end of the file and the number of lines to read, n, refers to that number of lines before (and including) line m.

See: COMMAND GETFILE

GOSHOW see SHOW

HELP

```
HELP <anything>
```

Does the same as TELL TELL. That is, edits a file that describes how to use the on-line documentation facility of P-EDIT.

See: MACRO TELL

HEX

```
HEX <value>
```

No value specified: Inserts below the current line the hex representation of the current line. The hex data may be removed by UNDO or DELETE.

Value specified: Displays the possible interpretations of value as character, hex and decimal. If value is not a hex or decimal number, it must be a single character.

HIDE

```
HIDE
```

Changes the definition of the current view so that it excludes the current line.

```
See: CONCEPT PARAMETRIC FILES
     MACRO   UNHIDE               (SIMILAR COMMAND TO GET LINE IN VIEW)
```

HIST

```
HIST <text>
```

The text line is appended to the HIST type file whose name is that of the current file being edited.

If issued with no argument, HIST file for current file is edited.

For use with MAINTAIN.

HUFF

```
<-> HUFF <disk> <tag>
           *      n
                  *
```

The file identified in each line of the specified range (in LIST file format) is HUFFed onto the specified disk or the same disk if '*' or unspecified. The file is updated to reflect the change.

If the range is specified by a tag of only one letter, the disk must be specified to avoid ambiguity.

IE

```
<-> IE <tag>
        n
        *
```

Deletes the lines in the specified range and executes them as P-EDIT commands (by stacking them). The range may contain no more than 100 lines. When the first of the commands is executed, the current line will be the line that was before it.

IF

```
<-> IF / cmmnd1 / cmmnd2 </ cmmnd3 </>>
```

Executes the first specified command. If there is no error, the next command is executed, if there is an error the third command, if any, is executed. The separator character (shown here as '/') may be any character not used in the commands.

The following command, perhaps assigned to the X command, will find the next occurence of 'A' or 'B', which ever comes first:

```
          IF /;TAG T1;L'A'/;TAG T2;GO T1;IF "L'B'""GO T2/GO T2
```

INCHANGF

```
<->INCHange /string1</string2</>>
   INCHX
   INCHY
```

Changes all occurences of the first specified string to the second from the current line on under interactive control. As each occurrence is found, a window around it is displayed with the found string underlined and the user is prompted for one of the following replies:

YES:	Make the change and find the next occurence.
NO:	Do not make the change and find the next occurence.
MAYBE:	Make the change but do not go on, but offer to unmake the change.
QUIT:	Stop the operation making no additional changes.

See:	COMMAND CASE	(Controls the handling of lower-case letters)
	COMMAND IMAGE	(Controls the handling of back-space and tab)
	COMMAND WINDOW	(Controls the which piece of the found line is shown)
	COMMAND ZONE	(Controls the part of the lines considered)

The standard SETUP P-EXEC defines the necessary synonyms and PF-key variable. If issued with a synonym ending in 'X', INCHANGE will set the X P-EDIT abbreviation so that issuing the command 'X' will continue the operation. If issued with a synonym ending with 'Y', the Y abbreviation will be set.

See: COMMAND X

INCHX see INCHANGE

INCHY see INCHANGE

INDENT

```
<-> INDENT <text>
```

Text specified: Inserts the text after the current line with the text indented to line up with the first non-blank character of the current line. More indentation can be specified by preceding the text with blanks (one extra is needed to separate the text from the command name). Less indenting can be specified by preceding the text with back-space characters.

No argument specified: Enters INDENTED INPUT mode where all further input is inserted into the file lined up with the previous line. Blank and back-space can be used as above.

COMMAND CASE	(Controls the handling of lower-case letters)
COMMAND DESERIALIZE	(Controls serialization field to be cleared)
COMMAND IMAGE	(Controls the handling of back-space and tab)
COMMAND TRUNC	(Controls the longest line that can be inserted)
MACRO REINPUT	(Permits P-EDIT commands to be entered in INPUT mode)

meaning of "back-space character"

Normally, a back-space character is as in EBCDIC (hexadecimal 16). The P-EDIT variable, INDENT.BACKSPACE, can be set to an alternative character string. Since back-space characters are looked for before blanks, a line beginning with a back-space can be inserted, indented, with the command:

```
INDENT @ @This line begins with one back-space
```

The standard SETUP P-EXEC defines the necessary synonym and establishes the input translation tables so that '@' will be translated to the EBCDIC back-space character.

INFILLIN

<->INFILLIN

INFILLIN allows one to interactively fillin any or all fill-in fields. As each fill-in field located, it is displayed and the user is prompted for one of the replies DEFAULT, NO, QUIT or a replacement string. If the reply is a replacement string, that fillin-field is replaced with the string. If the reply is DEFAULT, the left and right brackets are removed from the fill-in field, leaving the default value. If the reply is NO, no change is made. In any of these cases, the next fill-in field is located and the user is prompted again. If the reply is QUIT, the macro terminates with no further action.

If a fill-in field has the same default value as one already filled in, the previous value will be used.

The pfkeys specified by the P-EDIT variable, INFILLIN.PFKEYS, will be set to enter the DEFAULT, NO and QUIT replies.
INFILLIN (like CHANGE) respects the zone settings and allows any delimiter.

related TELLs

MACRO FILLIN	(Defines the concept of fill-in filed)
MACRO FIELDBRACKETS	(Redefines the strings used to bracket fill-in fields)

INFIX

<->INFIX

Interactively sets a mask to fix (or exclude) all unfixed lines of the current file. As each unfixed line is found, it is printed, if hidden, and the user is prompted for one of the following replies:

FIX: Restrict the default mask so that the line becomes fixed.

EXCLUDE: Restrict the default mask so that the line becomes excluded.

CONTINUE: Come back to this line later.

QUIT: Stop processing with mask as already set so far.

If the operation completes by running out of unfixed lines (rather than QUIT), the file type is set for a fixed file (if the P-EDIT variable, FIXED.FILE.-TYPE.fnumber is set) and the format is set to be non-parametric.

See: CONCEPT PARAMETRIC FILES

The standard SETUP P-EXEC defines the P-EDIT variable, INFIX.PFKEYS, so that PF-keys can be used for user responses. The standard MODEL P-SETUP defines the P-EDIT variable, FIXED.FILE.TYPE.fnumber (where fnumber is the file number of the current to the appropriate file type.

INSPELL see SPELL

INSTANCE

INSTANCE name

Creates a file of type SCRIPT and name as specified by inactively fixing the current file. The current file must be of type FORM. That file is then edited and its fill-in fields are interactively filled in. It is then saved.

See: MACRO INFIX	(Used to fix file)
MACRO INFILLIN	(Used to fill in fill-in fields)

INVERT

```
<-> INVERT <tag>
           *
           n
```

Inverts the order of the lines in the specified range. If no range is specified, two lines is assumed.

J see SUBST

K

```
K  /string/ <indentation>
KC    n
KR
KT
```

The operation descibed below is done up to column specified. It can be specified as a column number or as a delimited string. The delimiter (shown here as a '/') may be any non-numeric, non-blank character not in the string. The column designated is the first of the first occurence of that string. When done and if specified by a number, the then current line is indented by that amount.

K: Splits the current line at the column. The second part becomes the current line and it is lined up with the first word of the first part.

KC: Combines the current and next lines by overlaying the second at the column. If no column is specified it is taken to be one after the last word on the first line.

KR: Removes columns 1 through the column on the current line.

KT: Truncates the current line at the column.

See:	COMMAND CASE	(Controls handling of case in column specification)
	COMMAND DESERIALIZE	(Controls serializtion field to be cleared)
	COMMAND IMAGE	(Controls handling of back-space and tab)
	COMMAND TRUNC	(Controls part of the line that will remain)

KC see K

KR see K

KT see K

LAND

```
<-> LAND /string1</string2 ... ></>
```

Goes to the next occurence of a line after the current line containing all of the specified strings. The delimiter (shown here as a '/') may be any non-blank character not in any of the strings. LAND will run somewhat faster if the most likely strings are specified early. Two strings in a line can overlap; thus LAND /ABC/BCD/ will locate the string 'ABCD'.

```
See: COMMAND CASE       (Controls handling of lower-case letters)
     COMMAND ZONE       (Controls the part of the lines searched)
     MACRO   AVAND      (Locate next line missing any specified string)
     MACRO   LOR        (Locate next line with any specified strings)
     MACRO   AVOR       (Locate next line missing all specified strings)
```

LAST see RANGE

LCASE

```
<-> LCase </string1</string2>/> <tag>
                                  n
                                  *
```

No strings specified: Converts the lines in the specified range to lower-case.

One string specified: Converts the first word starting with that string to lower-case on each line of the specified range.

Two strings specified: Converts the first string that starts with the first specified string and ends with the second to lower-case on each line of the specified range.

```
<-> UCase </string1</string2>/> <tag>
                                  n
                                  *
```

Same as LCASE, except the conversion is to upper case.

The separator (shown here as '/') may be any special character not in the strings except an '*' or blank.

```
See: COMMAND DESERIALIZE     (Controls serialization field to be cleared)
     COMMAND IMAGE           (Controls the handling of back-space and tab)
     COMMAND ZONE            (Controls the part of the lines considered)
```

The standard SETUP P-EXEC defines the necessary synonyms.

LEFT see RIGHT

LF see FILELIST

LHEX

```
<-> LHEX </>hex</> <tag>
                    n
                    *
```

Like LOCATE, except the string to be located is expressed as the hexidecimal (EBCDIC). The separator character, shown as '/', may be any non-blank character that is not a hexidecimal digit. Since it serves no purpose, other than to make the analogy with LOCATE stronger, it may be ommitted.

LINKS

```
LINKS <vaddr>
```

Virtual disk specified: Displays all current links to the disk in the message area.

No disk specified: Picks up the virtual disk address from the second word of the current line (the format produced by DISKS) and inserts comment lines that show the current links.

```
See: MACRO DISKS      (Creates a file of accessed disks)
```

LISTFILE see FILELIST

LM see FILELIST

LMCASE see MCASE

LONGEST

```
<-> LONGEST
```

Goes to the longest line in the file. If there is more than one line whose length is equal to the longest, goes to the next one following the current line or, if none, the first one in the file.

LOR

```
<-> LOR /string1</string2 ... ></>
```

Goes to the next occurence of a line after the current line containing any of the specified strings. The delimitor (shown here as a '/') may be any non-blank character not in any of the strings. LOR will run somewhat faster if the most likely strings are specified early.

```
See: COMMAND CASE     (Controls handling of lower-case letters)
     COMMAND ZONE     (Controls the part of the lines searched)
     MACRO   AVOR     (Locate next line missing all specified strings)
     MACRO   LAND     (Locate next line with all specified strings)
     MACRO   AVAND    (Locate next li.. missing any specified string)
```

LT see FILELIST

LUF

```
LUF /string</>
```

Locate the unique occurrence of the specified string in the current file. 'Not found' or 'Not unique' is reported if so (return code of '1' for macros).

The delimiter, shown here as '/', may be any non-blank character not in the string.

See: COMMAND	CASE	(Controls the handling of lower-case letters)
COMMAND	ZONE	(Controls the part of the lines considered)
MACRO	CUF	(Analogous CHANGE command)
MACRO	LUS	(Locates unigue string on screen)

LUS

```
LUS /string</>
```

Locate the unique occurence of the specified string on the screen. 'Not found' or 'Not unique' is reported if so (return code of '1' for macros). If SCOPE is OFF, 'Not found' is reported.

The delimiter, shown here as '/', may be any non-blank character not in the string.

Characters not being displayed because they are lost due to lines being displayed bright, are still considered to be "on the screen".

See: COMMAND	CASE	(Controls the handling of lower-case letters)
COMMAND	SCOPE	(Controls whether SCOPE is ON)
COMMAND	WINDOW	(Controls the part of the lines considered)
MACPO	CUF	(Analogous CHANGE command)
MACRO	LUF	(Locates unigue string in file)

MAKESAME

```
MAKESAME <file>
```

Modifies the current file in the "minimum" fashion to make it the same as the file with the specified file number (or the previous file in ring, if none). The two files must either by non-parametric or have sufficient masks set to entirely fix them. MAKESAME uses the executable module, QCOMPARE 370, to find the "minimum" differences.

See: CONCEPT PARAMETRIC FILES

example

MAKESAME permits two files that share a common origin to be merged into one parametric file that represents both of them. Thus it is both a way to get started using the parametric facility of P-EDIT and to integrate the modifications made by those who don't with a parametric file under development. A sequence of commands like the following can be used to merge two files into one that represents them both:

```
P-EDIT FIRST FILE
P-EDIT SECOND MODEL
(Respond with 'FILE', the file type in this example, when prompted)
GET SECOND FILE
MASK FILE=FIRST
MAKESAME
UNMASK
```

MCASE

```
<-> MCase command
```

Issues the specified command with mixed case set temporarily on (CASE M M). The command name may be specified as a prefix on MCASE by defining the appropriate synonym.

```
<-> CMCase </string1/string2/ <n <g>>>>>>
```

The same as the built in CHANGE command except temporarily imposes mixed case.

```
<-> LMCase /string/
```

The same as the built in LOCATE command except temporarily imposes mixed case.

See: COMMAND CASE   (Controls the treatment of lower-case characters)

The standard SETUP P-EXEC defines the necessary synonyms.

MORE

```
MORE <i <j>>
```

Displays the lines around the current line in a COMPARE output file by going and reading the appropreate CMS files. The positive numbers i and j specify the left and right margins for the display. The maximum window is 78. Default values are: i = 1, j = i + 77.

See: MACRO COMPARE   (Produces a COMPARE output file)

MOVE

```
<-> MOve <<tag1>-tag2> <To tag3>
    COpy       n         Up m
               *         Up *
                        Down m
                        Down *
                         Here
```

Moves (copies) the lines in the specified range to the specified destination The range may be specified by a line count in the normal fashion, by two tags (the first will default to the current line but the '-' must be coded), or by prior use of FIRST and LAST. The destination may be specified relative to the current line (UP, DOWN or HERE), by a tag (TO), or by prior use of TO. If FIRST, LAST or TO are used, all default tags are cleared.

The range may be in a different file than the destination. However, in the case of MOVE, any tags on the moved lines will not move with them as normal (they remain on the now deleted original lines).

See: MACRO RANGE   (Used to set default tags: FIRST, LAST and TO)

The standard SETUP P-EXEC defines the necessary synonyms.

NOTE

NOTE <text>

This macro is used to leave short-term notes containing a location in the file and some text (often a P-EDIT command). If text is specified, it is remembered along with the current line. If not, the earliest note is put into the input area and cleared.

ORDER

```
<-> ORDER <tag>
           n
           *
```

Re-ORDERs (in the CP sense) the spool files designated by the lines in the specified range (in CP-LIST format). That is they are made the next files to be processed in their respective devices. Lines are not moved to reflect the new order, however, lines designating files that do not exist are modified accordingly. The current line will be the last line of the range (or, if a line with improper format is found, that line).

See: MACRO CPLIST   (Generates CP-LIST files)

OUT

OUT

Quits editing all files except those that have not been saved and are not dummy files (have an '*' as their file name, type or mode). Edits a file of the remaining files or leaves P-EDIT if none.

```
See: MACRO QUIT       (Quits editing one file)
     MACRO CANCEL     (Quits editing all files with no protection the unsaved)
```

P-EDIT

```
P-Edit <fn <ft <fm <form> <width>>>> <(options>
Erf
```

Arguments specified: Performs the same operation as P-EDIT EXEC unless the file specified is unique and is already being edited, in which case it is resumed.

No arguments: If the current file is of one of the following types, a file name will be picked up from the current line if it is in the expected format and the file will be edited or resumed. Otherwise, a file of type P-LIST describing all files being edited is generated and edited.

```
LIST:     MACRO FILELIST      (Generates lists of files on disk)
CP-LIST:  MACRO CPLIST        (Generates lists of spool files)
P-LIST:   MACRO P-LIST        (Generates lists of files being edited)
TELL:     MACRO TELL          (Generates TELL files with cross-references)
MAIL:                         (MAIL files might have cross-references)
```

In any case, if more than one copy of the file is already being edited, a a file of type P-LIST describing them

```
See: CONCEPT INITIALIZATION      (How initialization by file type is done)
     MACRO   QUIT                (Stops editing the current file)
     MACRO   FILE                (Saves and then stops editing the current file)
```

The standard SETUP P-EXEC defines the necessary synonyms.

P-LIST

```
<-> P-LIST <tag>
           n
           *
```

No arguments: Edits a file that has a line for each file that is in the ring of files being edited.

Argument: Reads all the files designated by the lines in the specified range (in LIST file format) into the ring of files being edited if they are not already there. A file is then edited, a subset of the above file, that has a line for each corresponding file in the ring.

This macro may be aborted by entering 'HE'.

```
See: MACRO P-EDIT       (Will resume the file described on the current line)
```

P-SCROLL

```
<-> P-SCROLL <type>
```

Goes to the next reasonable piece of the file according to file type. If no file type is specified, the file type of the current file is used.

File types:

TELL: Goes to the line that will center the next page of documentation on the screen (if SCOPE is OFF, prints the next page of documentation).
```
     See: MACRO TELL       (Produces the pre-formatted TELL type files)
```

LISP: Goes to the next start of a LISP function if one can be found close enough for some text to overlap between the text displayed before and after; otherwise, the same as SCROLL. The test for the start of a LISP function is a left parenthesis in column 1 or 2.

Any other: Does the same as SCROLL.
```
     See: COMMAND SCROLL
```

P-SCROLL is intended to be used as a PF-key assignment by P-SETUP files. They look in the variable, SCROLL.PFKEYS, to find the proper keys.

See: CONCEPT INITIALIZATION

PARA

```
PARA <n <p <q>>> <(<Indent> <Cap> <No.><)>>
      *  *  *
      B
```

Formats the lines in the specified range as a single paragraph that extends from column 1 (or or p) to column 70 (or q). If the range is specified as 'B', the lines from the current line to the next blank line will be formatted. Words will in general be separated by a single space. An extra space will be inserted at the end of each sentence. The following options are permitted:

```
INDENT: Indent the first line by five additional spaces.
CAP:    Capitalize the first letter of each sentence.
NO.:    Treat '.' as any other character.
```

```
See: COMMAND DESERIALIZE      (Controls serialization field to be cleared)
     COMMAND TRUNC            (Controls part of line used and maximum produced)
```

use of special variables

By setting certain P-EDIT variables, the behavior of PARA can be modified.

PARA.DEFAULTS: Up to three values to be used as default values for the first three arguments to PARA. If the variable, PARA.DEFAULTS.type (where "type" is the file type of the current file), is found, it is used instead. If fewer than three values are found, the missing ones are taken to be 1, 1 or 70, respectively.

PARA.LINE: PARA normally leaves the current line at the last line in the paragraph. If this variable has one of the following values, this action is modified:

```
FIRST: Leave current line at the first line of the paragraph.
NEXT:  Leave current line at the line after the paragraph or, if it is
       blank, the line after that.
```

```
See: CONCEPT VARIABLES          (How to set P-EDIT variables)
     CONCEPT INITIALIZATION     (How to get them set automatically)
```

PARENTHESES see BALANCE

PASTE

```
<->PASTE
```

Appends the current line to the end of the previous line.

```
See: MACRO    CUT       (Opposite of PASTE)
     RESOLVE KC         (Similar to PASTE)
```

PAUSE

```
PAUSE
```

Does the following time consuming clean-up operations that are appropriate when the user feels that he will not want to interact with P-EDIT for a while. This is a good candidate for a PF-key.

1- All files that have been modified since they were last saved are saved.

2- Storage reclamation is done if less than 100,000 bytes are available.

See: COMMAND SAVE
     CONCEPT RECLAMATION

PEDIT

```
PEDIT fn ft <fm <format <width>>> <( <Dummy> <Nodef> <Quiet> <DARK> <)>>
             *
```

Invokes the PEDIT editor while remaining in P-EDIT. Primarily used to compare features in these two very similar editors.

```
See: COMMAND VERSION      (Can be used in either editor to print which)
     COMMAND EDIT         (For details of arguments (NODEF is like NOSETUP))
```

PRESUME

```
PRESUME <prefix>
        OFF
```

Prefix specified: Enters a mode where all further lines typed in until another PRESUME command will be prefixed by the specified text unless the line is null or ends with a blank (for special treatment of named interrupts and immediate commands see below). For convenience, if the specified text does not end in a blank, one will be provided unless it ends with a back-space (often typed as '@') which will be removed. Three useful examples are:

```
PRESUME INSERT         (Defaults to insert lines into file)
PRESUME INDENT         (Defaults to inserting and lining up lines)
PRESUME CHANGE //@     (Defaults to placing text at beginning of ZONE)
```

OFF specifed: Clears the presumption mode.

Nothing specified: Prints (stacks) the current presumption.

Note -- Since immediate CMS commands and named interrupts (like HX and HC) will be interpreted by CMS unless they are followed by a blank, there would be no way to get the presumption prefixed to such a line unless they were treated specially. Therefore the blank will be ignored on such a line. In the very unlikely event that a macro has been given the same name, it can be called by following its name with two blanks.

The standard SETUP P-EXEC defines PRESUME.HEADER.NAME as a variable to look to for the header. When thus defined, the current presumption is displayed in the header.

PROMOTE

```
PROMOTE
```

The current line is forced to be in all versions of the file consistent with the current masks. That is, its Boolean expression is changed to be its old value ORed with the AND of the current masks.

PUFF

```
<-> PUFF <disk> <tag>
          *      n
                 *
```

The file identified in each line of the specified range (in LIST file format) is PUFFed onto the specified disk or the same disk if '*' or unspecified. The file is updated to reflect the change.

If the range is specified by a tag of only one letter, the disk must be specified to avoid ambiguity.

PURGE

```
PURGE relation
```

Makes the current file less parametric. Any version under control of the specified relation will be deleted. Examples:

```
PURGE TIME<79                 Remove history before 1979
PURGE TIME>=79.1.2            Back up to file as of January 1, 1979
PURGE TIME<CURRENT(TIME)      Purge all history
PURGE OPTION=1                Get rid of option 1
PURGE OPTION¬=1               Make option 1 the only case
```

If the intention is to reduce the size of the phycical file, take care to reset all masks (such as the TIME mask), since the deletion is done under their control as usual.

PUSHPOP

```
PFKEY n IMM PUSHPOP
```

Intended for use by a PF-key. If the PF-key is struck once, same as PUSHMASK; if struck twice, same as POPMASK. In reality, if the current default mask is not TRUE (which would indicate it had not been set since previous PUSHMASK) a PUSHMASK is executed; if it is TRUE, two POPMASKs are executed.

The PF-key should end with a blank (not shown above) so that PRESUME will recognize it as a command.

See: COMMAND PUSHMASK

PUTBOOL

```
PUTBOOL name fn<.<ft<.fm>>> <boolean>
```

Appends the Boolean expression (with functions expanded) to the specified file associated with the specified name. If the file already has a Boolean expression associated with the name, it is deleted. If no Boolean expression is specified, any existing definition is deleted. If the file type is not specified, 'P-BOOLS' is assumed; if the file mode is not specified, the mode of the current file is assumed. The file name must be a legal parameter value as as specified.

```
See: CONCEPT PARAMETRIC FILES
     FUNCTION READ                (Retrieves Boolean expression from file)
     FUNCTION MASK                (Can be used to save a mask value)
     FUNCTION FETCH               (Can be used to save an edit variable value)
     FUNCTION MASKS               (Can be used to save the AND of current masks)
```

QCOPY

```
<-> QCOPY <disk> <tag>
                 n
                 *
```

Copies the CMS files designated on each line in the specified range (in LIST file format) onto the specified disk (or the A-disk, if none is specified). Inserts a comment line describing the new files after each line corresponding to the original file.

If the range is specified by a tag of only one letter, the disk must be specified to avoid ambiguity.

```
See: MACRO FILELIST      (Generates LIST files)
```

QMOVE

```
<-> QMOVE <disk> <tag>
                 n
                 *
```

Moves the CMS files designated on each line in the specified range (in LIST file format) onto the specified disk (or the A-disk, if none is specified). Updates each line to it describes the new file. If the original disk is not accessed read/write, HIDE is used to simulate the move.

If the range is specified by a tag of only one letter, the disk must be specified to avoid ambiguity.

```
See: MACRO FILELIST      (Generates LIST files)
```

QUIT

```
Quit
```

Stops editing the current file and returns to CMS if it was the only file being edited. Unless one of the following conditions is met, a warning is given that an unsaved file has been QUITed (if it was the only active file, the return to CMS will be inhibited):

```
     Most recent change has been saved
     File's name, type, or mode is '*' (a dummy file)
```

```
See: MACRO FILE          (Saves file on disk and then quits)
     MACRO OUT           (Quits all files that have been saved)
     MACRO CANCEL        (Quits all files no matter what)
```

The standard SETUP P-EXEC defines the necessary synonym.

RANGE

```
<-> FIRST <Cancel>
    LAST
    TO
```

No argument specified: Sets the specified default tag to the current line unless it is unfixed. If it is unfixed, it must be the first (or last for LAST) unfixed line in the view among a consecutive group of unfixed lines; the default tag will be set to the first (last) unfixed line in the group.

CANCEL specified: Clears the specified default tag.

```
<-> RANGE  <n>
           Cancel
```

No argument specified: Same as FIRST if not yet set, otherwise same as LAST if not yet set. If both set, they are cleared. This is a good candidate for a PF-key.

Number specified: Same as a FIRST issued on the current line and a LAST issued on the line n-1 lines forward.

CANCEL specified: Clears the default tags, FIRST and LAST.

The standard SETUP P-EXEC defines the necessary synonyms. If used with a synonym that does not begin with 'F', 'L', or 'R', TO will be assumed.

These define the range and destination of several macros. Use of these default tags will clear them.

```
See: MACRO MOVE
     MACRO COPY
     MACRO SORT
```

REACCESS

```
REACCESS
```

Re-accesses the disk designated by the current line. If the disk was accessed read-only, it is re-accessed with the NONSHARE option.

Note -- This command only re-accesses disks that are still linked to the virtual machine, it will not re-link the disk as the CMS command, GIME, will.

```
See: MACRO DISKS     (Generates a list of accessed disks)
```

RECEDIT

```
RECEDIT <fn <ft <fm <format> <width>>>> <(<Dummy> <Quiet> <Nosetup> <)>>
```

Calls P-EDIT recursively; otherwise just like the EDIT command. This is for use by P-EDIT macros and EXEC's. If P-EDIT is your normal editor, NORMAL EDITOR should contain the line: P-EDIT P-EDIT RECEDIT.

REINPUT

```
<-> REINPUT <command>
```

Executes the specified command and re-enters INPUT mode if the previous command had entered INPUT mode. Primarily for use by PF-keys: PFKEY 03 IMM #REINPUT IE.

If no command is specified, it is taken as the next line read. Thus, the following PF-key will "prime" P-EDIT to take the next line, only, as a command even though it is in INPUT mode: PFKEY 03 IMM #REINPUT.

The prior existence of input mode is determined by looking at the header line on the screen converted to upper case. If it is <REVERSE> INPUT: or <REVERSE> REPLACE: the INPUT or REPLACE command is use to re-enter INPUT mode; if it is <REVERSE> macro INPUT<:> the macro named is used to re-enter INPUT mode.

Commands that support various kinds of INPUT mode:

```
     COMMAND INPUT      (Normal INPUT mode)
     MACRO   INDENT     (Lines up lines in structured files)
     MACRO   BLIND      (Only refreshes sceen on request)
```

RENAME

```
<-> RENAME fn <ft <fm>>
           *   *   *
           =   =   =
```

Renames the CMS file designated on the current line (in LIST file format) to the name specified. Parts of the file name that are missing, '*' or '=' will remain unchanged. The current line is updated to reflect the change.

If the disk is accessed read-only, the file directory will be modified to make the change temporarily.

```
See: MACRO FILELIST     (Generates LIST files)
```

RENEW

```
RENEW <file>
```

No argument specified: Updates the current line (in LIST file format) to the current information about the file named. If more than one file is found, lines will be inserted for the excess files. If the file no longer exists, the line is changed to a comment line. Comment lines and lines with garbage cause no effect.

File specified: Inserts a line for each file specified. The files are specified as a FILELIST pattern.

```
See: MACRO FILELIST     (Similar command)
     MACRO FOR          (Used if a number of lines are to be updated)
```

REPLAY

```
REPLAY <n>
```

Similar to UNDO. The specified number of commands are undone and each, in turn is put into the input area. The commands can be modified however the user wiches included using CP's line-end character (often '#') to execute more than one command. When ENTER is pushed, REPLAY will execute the commands in the input area, and will put the next undone command in it.

The maximum number of commands that can be REPLAYed is two less then the number established with the REMEMBER command; therefore the default maximum is seven.

RESERIAL

```
RESERIAL <incr>
```

The lines around the current line that have blank serialization fields will be serialized with values between the serialization values on the two lines before and after them. The increment specified will be used or, if none, a reasonable one will be used. It is computed by finding the power of ten less than the difference between the two bracketing serialization fields. If the bracketing serialization fields have alphabetic headers, these will be honoured but must be equal.

The current position in the file will be unchanged unless the increment is too large to fill in all the blank serialization fields. In that case, the line after the last one reserialized will become the current line (return code of 1 for macros).

See: COMMAND DESERIALIZE   (Controls serialization field to be used)

RETCODE

```
RETCODE command
```

Issues the command, prints its return code, and then undoes all effects of the command (in so far as possible). Primarily used during the process of writing macros in order to test which return code the macro would get if it issued the command.

See: CONCEPT MEMORY   (Which effect of the command can be undone)
     CONCEPT MACROS   (How to write macros)

RETROFIT

```
RETROFIT <fnumber>
```

Modifies the current file so that it includes the modifications made parametrically to the specified file. The intention is that the two files were at one time the same and were modified independently. The algorithm looks for the unique and fixed occurence in the current file of the view of each unfixed section in the specified file. If found, it makes the change to the current file; if not, the change is put at the top of the current file for manual handling. Because the view is used to trigger a match, it should be adjusted to be as much like the current file as possible.

If no file is specified, the default of the previous file is taken. That file must be parametric. If the current file is not parametric, it is made so and the file type is changed to *. At the end of the operation, the two files will share the same control. A return code of 1 indicates that some modifications were not found. A return code of 2 indicates some other problem; no change has been made.

Typing ?? during operation will type an indication of how much has been done.

REVEAL

```
REVEAL /string</>
       command
         OFF
```

String specified with '/': Enters REVEAL mode with every line with the string revealed.

Another command specified: Enters REVEAL mode with every line that "satisfies" the command revealed. The command should be in the "LOCATE family", that is, a command the searches for something starting with the next line and returns 0 or 1 depending on whether it is found. A line "satisfies" the command if it returns a 0 with it as the current line. Since the command is executed at different points in the file, it should not have a range specified as a line count.

OFF specified: Leaves REVEAL mode. This must be done before saving the file on disk.

REVEAL uses the parametric feature of P-EDIT to create a version of the file consisting of only the lines that satisfy the specification. P-EDIT commands will operate on both the full file and the short version thus created. However, only the short version will be seen. Lines of the short version will be highlighted where a line in the full file is not being displayed. While in REVEAL mode, the file mode of the current file is changed to '*' to discourage writing it on disk.

Note -- REVEAL assumes that the lines to be revealed constitute a small percentage of the total lines. If this is not true and the current file is large, it will take an excessive length of time to complete.

```
See: CONCEPT PARAMETRIC FILES
     COMMAND HILITE              (Controls the method of highlighting)
     MACRO   UNHIDE              (Can be used to see the full file)
```

RIGHT

```
RIGHT <n>
>      *
LEFT
<
```

The current definition of WINDOW is moved right (left) by the specified amount with the width of the WINDOW kept constant. If no amount is specified, the width of the WINDOW is assumed. If '*' is specified, the maximum amount is assumed. If the amount specified goes beyond the current WIDTH of the file, it is lessened so the WINDOW is the first (last) whatever the WINDOW width is.

SAME

```
<-> SAME <Name> <Type> <Disk>
```

Inserts lines describing all files with the same file name, type, and/or disk (as specified) as the file designated by the current line (in LIST file format). If no argument is specified, NAME and TYPE are assumed.

If the current line is null or begins with an '*' or there no other file is found, nothing is done.

See: MACRO FILELIST   (Generates LIST type files)

SAMPLE

This is intended to be a good base for writing macros. It has the following facilities:

```
&CALL -EXIT <retcode <message>>
```

If macro was called by user, warning message is issued; if called by macro, retcode is returned. If retcode = 0 any message will be printed without warning.

```
&IRREVOCABLE = YES
```

After this, any interrupts (no storage or user) will warn the user that the macro was partially executed. This is important after issuing a CMS command that modifies tha file system, say.

```
&MUST     = &STRING OF command
```

It [illegible] times crucial that some P-EDIT command is executed even if an in[illegible] occurs. The value of &MUST.DO will be executed after interrupt clean-up.

Also contained in this sample macro is the logic necessary to parse and loop over a range of the form:

```
<tag>
  n
  *
```

SAVEFIX

```
SAVEFixed <fn <ft <fm <format> <width>>>>
           *   *   *
```

Unfixed lines in current file: Prompts the user for which version of the logical file he wishes to write on disk and then writes it as specified.

Logical file is already fixed: Simply writes the logical file on disk as specified.

The format, if specified, must be F, V, or VB. The file type, if not specified, will be the current file type unless the P-EDIT variable, FIXED.FILE.TYPE.fnumber, is set. If this variable is not set, some argument must be specified.

```
See: COMMAND SAVEVIEW            (Similar command for files already fixed)
     MACRO   INFIX               (Used to do the interactive fixing)
     CONCEPT PARAMETRIC FILE
```

The standard SETUP P-EXEC defines the necessary synonym. The standard MODEL P-SETUP defines FIXED.FILE.TYPE.fnumber (where fnumber is the file number of the current file) to the appropriate file type.

SAVEFIXED see SAVEFIX

SAVEFORM

```
SAVEFORM fn <ft <fm>>
```

This is to be issued while in a FORM file. FORM files are parametric files with fill-in fields. They typically start with SCRIPT comment lines describing the parameterization. SAVEFORM generates a SCRIPT file called name which contains only the SCRIPT comment lines and the fill-in fileds. The intention is that this would be filled out by someone for later entry by someone else into the system.

```
See: MACRO VERSION       (Used to generate one version from a FORM file)
     MACRO FILLIN        (Basic information about fill-in fields)
```

SCALE

```
<-> SCALE <width>
```

Inserts a scale line after the current line in the file. The scale line extends from column 1 through the specified width. If no width is specified, it is assumed to be 80.

```
See: CONCEPT SCREEN FORMAT
```

SEARCH

```
<-> SEARCH n <command>
           *
```

The files designated by the current and next n-1 lines of the LIST of files currently being edited are searched for those files which "satisfy" the command. If a file is suspended, it will be resumed, otherwise edited anew, then the command (typically a LOCATE or FIND) is performed on the file. Whenever the command is satisfied, a new level of the editor is entered to read commands from the console (or stack). Upon EXITing (QUITing or FILEing) from that level, the search resumes. If the command is not satisfied, the file will be DROPped if it is saved and was not previously being edited, otherwise is is retained by the editor for later resumption.

A command is "satisfied" if, when executed starting at top-of-file, it does not give return code 1 (e.g. does not go to end-of-file). Star ('*') is valid for n and means all remaining lines of the LIST file.

The search may be stopped when done with the current file by entering the HE named interrupt.

For the purposes of UNDO and ?, the searching of each file is a separate command if the SEARCH command was issued from command level.

SETCOL

```
<-> SETCOL column /string</ <tag>>
                                *
                                n
```

Changes the characters starting from the specified column to the specified string over the specified range. The delimiter (shown here as '/') may be any non-blank character not in the string. The final delimiter may be ommited if the command is to operate on the current line.

This command is unusual in that it will do the change without regard to the text in the lines or file characteristics (such as WIDTH).

SETUP

```
SETUP <type>
```

Re-initializes the current file as it would have been initialized when editing of it began had it been of the file type specified (or the file type DEFAULT if not).

See: CONCEPT INITIALIZATION

SHORT

```
SHORT name <command>
```

Defines the specified name as a synonym for the X command so it can be used as a short name for commands just as X and Y are. If a command is specified, it is defined as the meaning of the name.

The SHORT command is also useful to assign a command to X or Y that those commands do not permit, such as SHORT X ?

See: COMMAND X

SHOW

```
<-> SHow      <tag>
    VIEWSHow   n
    VSHow      *
```

Starts a loop, under user control, to step through each version of the lines in the specified range. If no range is specified, all consecutive unfixed lines around the current line are assumed. Normally the stepping through is accomplished by setting a mask, however, if the optional prefix, 'V' or 'VIEW', is specified, the versions are stepped through by changing the view. A SHOW issued in the middle of another show loop acts reasonably.

```
STep <<+>n>
       -n
```

Advances the lastest SHOW loop as specified (or to next version, if not). A negative advance will go back to a previous version.

UNSHOW and GOSHOW

UNShow

Terminates the lastest SHOW loop. If it was a loop within a loop, the previous loop is resumed.

GOShow

Goes to the first line in the range of lines the lastest SHOW loop is stepping through the versions of.

See: CONCEPT PARAMETRIC FILES

The standard SETUP P-EXEC defines the neccesary synonyms.

SHOWMASK

SHOWMasks

Creates a file showing all mask names and their values for the current file. This file is then edited.

The standard SETUP P-EXEC defines the necessary synonym.

SHOWMASKS see SHOWMASK

SHOWPARM

```
<-> SHOWPArms <tag>
               n
               *
```

Edits a file showing all parameters with associated values for the lines in the specified range (or the entire file, if none). This operation is independent of any masks that are set (except for the meaning a numeric range specification).

See: CONCEPT PARAMETRIC FILES

The standard SETUP P-EXEC defines the necessary synonym.

SHOWPARMS see SHOWPARM

SHOWPF

SHOWPFkeys

Creates a file showing the current settings of pfkeys 1-12 in an array of the same form as the keys themselves: 4 rows and 3 columns. This file is then edited.

The standard SETUP P-EXEC defines the necessary synonym.

SHOWPFKEYS see SHOWPF

SHOWSYNONYMS see SHOWSYNS

SHOWSYNS

```
SHOWSYNS
SHOWSYNonyms
```

The current P-EDIT synonyms are put into a temporary file which is then edited. The file consists of two sets of columns, the first is sorted by command, the second by synonym. The synonym beyond its minimum length is displayed in lower case.

The standard SETUP P-EXEC defines the necessary sysnonym.

SHOWVARS

```
SHOWVars
```

Creates a file showing all edit variables and their values. This file is then edited.

The standard SETUP P-EXEC defines the necessary synonym.

SHOWVERS

```
SHOWVERsions
```

Creates and edits a file showing all versions of the current (parametric) file that are permitted by the current masks. This file begins with the Boolean expressions that define each version. This is followed by the lines in the logical file. From each Boolean expression, a line, formed with '|', is made running the the length of the file. The '|' is changed to an '*' for each line in the logical file corresponding to the versions it is in.

This macro is not really practical on very large files or files which have many versions consistent with the current masks). The header on the screen is modified after each version is handled.

The standard SETUP P-EXEC defines the necessary synonym.

SHOWVERSIONS see SHOWVERS

SIMEDIT

```
SIMEDIT ON
        OFF
```

SIMEDIT is a P-EDIT macro which provides an interface to P-EDIT which removes the distinction between the "Input" and "Edit" modes of operation.

Under SIMEDIT, any line entered from the terminal which is not a legal P-EDIT command is treated as input, thus obviating the need for a special input mode. (Note that the P-EDIT input mode is still supported.) This is useful in most cases where text entry and editing are performed simultaneously. It does require some awareness about unknown P-EDIT macros which may be executed when text entry was intended.

discussion of macro handling

Any standard P-EDIT command will be executed just as it would be under P-EDIT. Thus caution is required during text entry if not in input mode. (If a valid P-EDIT command is to be entered as text, it can of course always be preceded by I and a blank). But suppose you wish to enter "Add 1 to ...". There exists a macro called ADD, and without trapping, it would be executed, and unexpected things could happen to you. Trapping works as follows:

The first time you encounter a macro, you will be prompted to enter an indication of whether to execute the macro, input the line or ignore the line. If you select execution or input, you may also specify whether you wish this to occur just this time, for the duration of the SIMEDIT session, or for all SIMEDIT sessions in the future. You may also specify if that the macro only be so processed for this file type or for all file types (this is handy for macros that only make sense for certain file types).

NOTE: After trapping, the reply MUST come from the terminal (READ IMMEDIATE). This protects you from problems that could occur when "typing ahead" etc.

SIMEDIT records permanent knowledge of which macros are to be executed and which are not in files of type SIMEDIT (on your A-disk); the file LIST SIMEDIT, for example, records the actions to be taken for macros issued when editing a file of type LIST. The file DEFAULT SIMEDIT records this information for those macros which are to be processed the same for all types of files. The processing associated with a particular file type takes precedence over the default processing.

SORT

```
<-> SORT <tag-tag> <<+><key1> <<+><key2> ... >>
          tag       -        -
          n
          *
```

```
Where a key is : col1-col2 | AGe | CLass | DIst | FM | FN | FT |
                 LAbel | MOde | NAme | ORigin | SIze | SPoolid |TYpe
```

The lines in the specified range are sorted according to the keys. If more than one key is specified, the first is the most significant (major) the second is the next most significant down to the last key being the least significant. If no keys are specified, the lines are sorted in ascending (default) or descending order using as the key the entire field specified by the current ZONE settings.

A range is a specification of a first and last line to be sorted. If no first is specified, it defaults to the current line. The first is specified either by a range of the form tag-tag or previously with the FIRST macro and no range specification in the SORT itself. The last can be specified by a number, n, an asterisk, a tag, or previously with the LAST macro and no specification in the SORT itself. A number means n lines forward (or backward for -SORT) from the current line on. An asterisk means the end-of-file (or top-of-file). The last line may not be before (after) the current line.

The keys may be numerical or symbolic column designations. A numerical key consists of a column number followed by a - followed by a column number, optionally preceded by an + or -; embedded blanks are not permitted. The column numbers denote the first and last columns of the key field, and the leading + or - denotes an ascending (default) or descending sort on that key.

Symbolic keys are those listed on the lines above and denote the corresponding fields in the following file types. The key AGE sorts on the date and time fields in the opposite order as that specified. Symbolic keys may be abbreviated by their first two letters.

Filetype	Generator	Sympolic keys permitted
LIST:	MACRO FILELIST	(AGE FM FN FT LABEL MODE NAME ORIGIN SIZE TYPE)
CP-LIST:	MACRO CPLIST	(AGE CLASS DIST FN FT NAME ORIGIN SIZE SPOOLID TYPE)
ARCHIVES:	MACRO ARCV	(AGE FM FN FT LABEL MODE NAME SIZE TYPE)

Note -- If no range is explicitely specified and the first key is symbolic, an order ('+' or '-') must be specified.

SPELL

```
<-> SPELL <text>
          n
          *
```

The user is warned of any incorrectly spelled words in the text and possible corrections if known. If a range is specified, text is picked up from the current line on until an incorrectly spelled word is found. Single letter words are always considered correct.

SPELL uses a copy of PROOFERS dictionary made 3-15-80. It uses the following phonetic hash to determine possible corrections. SPELL will offer as possible corrections the words with the most triads matching.

SPELL can be terminated with the named interrupt, HE. The file WORDS P-TABLE can be used to provide a small number of additional recognized words

description of INSPELL

```
<-> INSPELL <Mixed> <Save> <MINimum n> <Display n>
```

Like SPELL * except whenever an incorrectly spelled word is found, it is shown underlined and the user is permitted to leave it as is, replace it with a specified correction, or replace it with one of the suggested corrections. If the file is mixed case and the original word is entirely capitalized, the correction will be forced to upper case. The response is remembered for the duration of the command and automatically applied if the same incorrect spelling occurs again unless that would cause truncation. If it occurs again with letters in different cases, it will be offered as a suggested correction.

INSPELL can be terminated with the named interrupt, HE. The file WORDS P-TABLE can be used to provide a small number of addition recognized words.

for meaning of options

MIXED:	Words in upper case will be considered correct.
SAVE:	Whenever the response to an error is a null line, that word is added to the file WORDS P-TABLE A. (If it is correct but you do not want it saved, a response of '0' can be used to change it to itself.)
MINIMUM n:	Words shorter than n in length will be considered correct.
DISPLAY n:	The screen will be redisplayed (or current line printed) at least every n lines.

for phonetic hash

1. Letters are converted to upper case and special characters are converted to blanks.

2. Letters are converted as follows: ABCDEFGHIJKLMNOPQRSTUVWXYZ
   EBCDEBCHECCLMMEBCRCDEBHCEC

3. All H's are converted to the previous letter unless at the beginning of a word or following a D.

4. Consecutive letters that are the same are reduced to one occurrence.

5. All E's are removed unless at the beginning of a word.

STEP see SHOW

SUBST

```
<-> SUBSTitute /string1</string2</string3</ <tag>>>
    J                                         n
                                              *
```

Changes the first occurrence of string designated by the first two specifiied strings to the third specified string on each line of the specified range. The designated strings are those that begin with the first specified string and end with the second, having any characters between them. Changed lines are recanonicalized if IMAGE is CANON. The last line scanned becomes the new current line.

The delimiter (shown here as '/') may be any non-blank character not in the strings and is optional if at the end of the command. Missing strings are taken as null.

examples of effects of null strings

SUBST //apple	(Deletes all characters up to and including "apple")
SUBST /	(Clears the current line (within the ZONE))
SUBST ///pear/4	(Replace 4 lines with "pear")
SUBST ///pear/4	(Replace the 4 lines with "pear")

See:	COMMAND CASE	(Controls handling of lower-case letters)
	COMMAND DESERIALIZE	(Controls serialization field to be cleared)
	COMMAND IMAGE	(Controls handling of back-space and tab)
	COMMAND ZONE	(Controls part of the lines considered)

The standard SETUP P-EXEC defines the necessary synonyms.

SUBSTITUTE see SUBST

SUM

```
<->SUM token <*>
              n
```

Displays or stacks the sum of the integers at the specified token position in the current line and the next n-1 lines.

TELL

```
TELL <<class> name>
```

P-EDIT is a self-documenting text editor. TELL is used to see some part of the documentation. If no class is specified, TELL will show the documentation for the command, macro or concept specified. For example, TELL N, will show the documentation for the NEXT command.

If no arguments are specified, TELL will attempt to pick them up from the current line (SCROLL twice for an example of such a line).

description of classes

The following classes can be specified:

Command:	Look for a P-EDIT command
Macro:	Look for a P-EDIT macro
Resolve:	Look for a command, then a macro, then a concept; this is the default
Exec:	Look for a P-EDIT EXEC (file type P-EXEC)
Setup:	Look for a P-EDIT setup EXEC (file type P-SETUP)
Function:	Look for a Boolean expression function (file type P-FUNC)
CONcept:	Look for a P-EDIT documented concept
Pfkey:	Look for a documented value for specified PF-key

P-MACRO, P-EXEC, P-SETUP and P-FUNC are acceptable synonyms for Macro, Exec, Setup and Function. In addition, they may be specified after the file name.

for list of P-EDIT command menus

CONCEPT BASIC COMMANDS	(Most commonly used commands)
CONCEPT BUILT-IN COMMANDS	(P-EDIT commands excluding macros)
CONCEPT CMS COMMANDS	(Interface to CMS and disks)
CONCEPT DISPLAY COMMANDS	(Control display or request information)
CONCEPT FILE COMMANDS	(Change file characteristics)
CONCEPT GLOBAL COMMANDS	(Change P-EDIT characteristics)
CONCEPT MODIFICATION COMMANDS	(Modify the text of files)
CONCEPT PARAMETRIC COMMANDS	(Deal with parametric files)
CONCEPT POSITION COMMANDS	(Redefine the current line)
CONCEPT RING COMMANDS	(Change which files are being edited)
CONCEPT OTHER COMMANDS	(Other commands)

for list of other documented concepts

CONCEPT CURRENT LINE
CONCEPT DICTIONARY
CONCEPT DIRECTION
CONCEPT IMAGING
CONCEPT INITIALIZATION
CONCEPT INTERRUPTS
CONCEPT MACROS
CONCEPT MAINTENANCE
CONCEPT MEMORY
CONCEPT PARAMETRIC FILES
CONCEPT SCREEN FORMAT
CONCEPT STATUS
CONCEPT STORAGE
CONCEPT SYNTAX
CONCEPT TAGS
CONCEPT VARIABLES

Boolean expression functions

FUNCTION CURRENT	(Get current value of name from disk, CMS or P-EDIT)
FUNCTION MEMBER	(Parameter equals one of set of values)
FUNCTION FETCH	(Gets value of edit variable)
FUNCTION READ	(Gets Boolean expression from disk)

TIME

TIME

Further restricts the TIME mask by TIME>=current GMT. If there is no TIME mask, sets one to TIME>=current GMT.

See: CONCEPT PARAMETRIC FILE

TO see RANGE

TRANSIN

```
TRANSIN <from to <back>>
          OFF
          CLEAR
```

Character translation specified: Whenever P-EDIT reads input, it will translate the first character to the second. Whenever P-EDIT writes the input area (as in the CHANGE command with no argument), it will translate the second character to the third. The third character should normally be the same as the first; this is the default. The characters may be specified as characters or as hexidecimal.

No arguments specified: The current input translations will be displayed or stacked.

OFF or CLEAR specified: All input translation is reset.

```
See: MACRO   TRANSOUT     (Controls output translation)
     COMMAND TRANSINX     (Built-in command needed to set input translation)
```

TRANSOUT

```
TRANSOUT <from to>
          OFF
          CLEAR
```

Character translation specified: Whenever P-EDIT writes output, it will translate the first character to the second. The characters may be specified as characters or as hexidecimal.

No arguments specified: The current output translations will be displayed or stacked.

OFF or CLEAR specified: All output translation is reset.

```
See: MACRO   TRANSIN      (Controls translatation of input)
     COMMAND TRANSOUX     (Built-in command needed to set output translation)
```

TRIM

```
<-> TRIM <* <length <ellipsis>>>
          n    *
         tag
```

The lines in the specified range (or current line if none) are shortened to the specified length (or the current TRUNC column) if longer. If an ellipsis is specified (often '...'), shortened lines will end with it; in this case lines, of course, more characters will be lost.

UCASE see LCASE

UNFIXED

```
<->UNFixed
```

Goes to the next group of unfixed lines. The earliest (without regard to any '-' prefix) of the found lines that are consistent with the current view becomes the current line. If they all are hidden, the line before them (without regard to any '-' prefix) becomes the current line.

```
See: CONCEPT PARAMETRIC FILES
     CONCEPT STATUS
```

The standart SETUP P-EXEC defines the necessary synonym.

UNHIDE

```
UNHIDE
```

Changes the definition of the current view so that it includes the current line.

```
See: CONCEPT PARAMETRIC FILES
     MACRO   HIDE                 (Similar command to remove line from view)
```

UNSERIAL

```
<-> UNSERIAL <tag>
               n
               *
```

Clears the serialization field in the lines in the specified range.

```
See: COMMAND DESERIALIZE     (Controls the serialization columns)
```

UNSHOW see SHOW

UNUPDT

```
UNUPDT <fn <ft <fm>>>
         *   *   *
```

Writes an UPDATE file with the specified name on disk. This command requires that the current file be parametric with the necessary masks set so that exactly two versions of the file are still in the logical file. The UPDATE file produced is such that were it applied to the hidden file, the viewed file would be produced.

Arguments that are missing or '*' default to cfn UPDATE cfm, where cfn and cfm are the name and mode of the current file.

UNUPDT can be used to communicate the latest version of a file to a remote location in the form of a "delta" from the previous version received by that location. This could be accomplished by a sequence of commands like:

```
PUSHMASK
MASK (VERSION=CURRENT(OTHER.LOCATION) | VERSION=CURRENT(VERSION))
VIEW VERSION=CURRENT(VERSION)
UNUPDT
POPMASK
```

```
See: CONCEPT  PARAMETRIC FILES
     FUNCTION CURRENT                 (Used to remember parameter values)
     MACRO    UPDATE                  (Will apply UPDATE files)
```

UPDATE

```
UPDATE <Fn <Ft <Fm>>>
        *   *   *
```

Applies the specified update file to the file currënt file. When done, the current line is at the last line inserted (or above the last deleted).

Arguments that are missing of '*' default to cfn UPDATE cfn, where cfn and cfm are the name and mode of the current file.

example

One use of UPDATE is as an aid in switching from using the UPDATE facility to maintain versions of a file to using P-EDIT's parametric file facility. This could be accomplished by a sequence of commands like:

```
P-EDIT PROGRAM MODEL
(Enter 'ASSEMBLE' when prompted)
GET PROGRAM ASSEMBLE
MASK VERSION>=UPDATE.1
UPDATE 1
MASK VERSION>=UPDATE.2
UPDATE 2
    ...
UNMASK
```

```
See: CONCEPT PARAMETRIC FILE
     MACRO   UNUPDT             (Used to regenerate UPDATE files when necessary)
```

VIEWSHOW see SHOW

VSHOW see SHOW

WHO

```
WHO
```

Prints information about the current user, machine and systems being used, and the current time and date.

Boolean Functions

CURRENT

```
CURRENT(name)
```

Files with called CURRENT P-TABLE are searched in the normal order to find name as the first word. If found, the second word on that line is returned. If it is not found, it must be USER, DATE, TIME, or FILE; in which case the current user id, current GMT date (yy.mm.dd), current GMT (yy.mm.dd.hh.mm.ss), or current file (name.type.mode) is returned.

FETCH

```
FETCH(name)
```

The value of the edit variable specified will be returned.

MASK

```
MASK(<name>)
```

The value of the mask name specified will be returned. If no mask name is specified, the current default mask name will be used.

MASKS

```
MASKS()
```

The AND of the current masks will be returned.

MEMBER

```
MEMBER(PARM, (VALUE1<, VALUE2<...>>))
```

Returns 'PARM=VALUE1 | PARM=VALUE2 ...' in order to test whether parameter is within set of values specified.

READ

```
READ(name, fn<.<ft<.fm>>>)
```

Reads the Boolean expression named from the specified file. If no file type is specified, P-BOOLS, is used; if no file mode, * is used.

Related TELL:
MACRO PUTBOOL

Other EXEC2 Files

ERROR

```
RESOLVORDER SYNONYM COMMAND+ MACRO+ ERROR
```

Edits the documentation (using TELL) of any command for which there is a syntax error. The above resolution order should be set during edit session initialization (say in PROFILE P-EXEC).

```
See: CONCEPT RESOLUTION
     CONCEPT INITIALIZATION
     CONCEPT SYNTAX
```

If a different treatment of syntax errors is desired, the file ERROR P-EXEC may be modified.

NUMBER

```
<-> n
```

Goes to the line being displayed the specified distance from line 10 on the display device (where the current line starts).

See: COMMAND SCROLL (The standard NUMBER P-EXEC is the same as SCROLL)

If a different interpretation of a number as a command is desired, the file NUMBER P-EXEC may be modified.

SETUP

Whenever the CMS command, P-EDIT, starts an edit session, the EXEC2 file, SETUP P-EXEC, is executed to initialize various values as follows. A standard SETUP P-EXEC is provided. Users may define their own if they wish to initialize in a different fashion, but this is not recommended for the typical user. Rather, the standard one provides various ways a user can specify his own initialization described below (PROFILE P-EXEC, TERMINAL P-EXEC and PFKEYS P-EXEC). The standard SETUP P-EXEC does the following unless the NOSETUP option has been specified in the P-EDIT command:

See: CONCEPT INITIALIZATION

Variables used for scope header: P-EDIT searches a specified list of edit variables to determine the left most field of the header line on the 3270 screen (the WHO field). A number of macros need a variable to be set for them if they are to effect the header. The standard definition is:

TEMP.HEADER.NAME	Used by macros that keep control
FIRST.LAST.TO	Used by RANGE (MOVE, COPY and SORT)
SHOW.HEADER.NAME	Used by SHOW (VSHOW)
PRESUME.HEADER.NAME	Used by PRESUME
SIMEDIT.HEADER.NAME	Used by SIMEDIT

Synonyms: Many macros need synonyms defined for them to work as documented (their TELL will say "The standard SETUP P-EXEC defines the necessary synonyms."). They define their needs with an *SYNONYMS line at the beginning of their P-MACRO file.

See: MACRO GETSYNS (Inserts the standard SYNONYM commands)

Terminal: The following is bypassed if a file named TERMINAL P-EXEC is found on the A-disk and it is executed instead. Terminal initialization consists of:

Setting SCOPE: The SCOPE mode is set to APL or NORM if the terminal being used is a 3270 (and the CMS switch, GRAFDEV, is ON). Otherwise, it is set OFF.

Setting VERIFY: If the SCOPE is set OFF, VERIFY mode is set ON.

Character translation: If the SCOPE is set ON, translation tables are setup to provide '¢' for tab (X05) and '@' for back-space (X16).

See:		
COMMAND	SCOPE	(Sets SCOPE mode)
COMMAND	VERIFY	(Sets VERIFY mode)
MACRO	TRANSIN	(Sets input translation)

PF-keys: The following is bypassed if a file named PFKEYS P-EXEC is found on the A-disk and it is executed instead. PF-key initialization consists of:

Defining PF-keys for certain macros:

INCHANGE.PFKEYS	PF02 PF05 PF08 PF11
INFILLIN.PFKEYS	PF02 PF05 PF08
INFIX.PFKEYS	PF02 PF05 PF08 PF11
INMERGE.PFKEYS	PF02 PF05 PF08 PF11
BLIND.PFKEY	PF11
SCROLL.PFKEYS	PF04 PF01

Setting PF-keys themselves (where '#' stands for 15 (hexidecimal) and '_' stands for a terminal blank):

Value	Meaning
01 IMM #COVERT -SCROLL _	(Display previous screen)
02 IMM #COVERT -UP _	(Go to previous line)
03 IMM #BACK _	(Undo another command)
04 IMM #COVERT SCROLL _	(Display next screen)
05 IMM #COVERT NEXT _	(Go to next line)
06 IMM #" _	(Do previous command again)
07 IMM #RANGE _	(Set current line as FIRST or LAST)
08 IMM #PUSHPOP _	(Once for PUSHMASK; twice for POPMASK)
09 IMM #REINPUT _	(Execute command in input area)
11 IMM #P-EDIT _	(Edit file named on current line)
12 RETRIEVE	(Retrieve latest line not yet retrieved)

If a PF-key is being used for tab, P-EDIT is told.

Additional initialization: At this point the file PROFILE P-EXEC is executed if found on the A-disk. There a user can place any other commands he wishes.

PF-keys

BACK

```
IMM #BACK
```

This PF-key undoes the effect of the last command. If used again, one more command will be undone, and so forth.

If P-EDIT was in INPUT mode, all lines inserted will be removed and it no longer will be in INPUT mode. The last the last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: CONCEPT MEMORY
     MACRO BACK

EXECUTE COMMAND

```
IMM #REINPUT
```

This PF-key is used to execute P-EDIT commands while in INPUT mode. To do so, enter the command, push the PF-key, and then push ENTER.

The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: MACRO REINPUT

INDENT

```
IMM #REINPUT CHANGE //  /
```

This PF-key is used to indent the current line. Indenting adds a double blank to the beginning of the line.

If P-EDIT was in INPUT mode, it will be restored after indenting the line. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

LISP SCROLL

```
IMM #COVERT P-SCROLL LISP
```

This PF-key moves the current line down to the last start of a LISP function definition if one can be found close enough for some text to overlap between the text displayed before and after. Otherwise, it moves the current line as much as possible to preserve some overlap. The test for the start of a LISP function is a left parenthesis in column 1 or 2.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: MACRO P-SCROLL

NEXT

IMM #NEXT

This PF-KEY moves the current line to the next line in the file.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: COMMAND NEXT

P-EDIT

IMM #P-EDIT

This PF-key is used to edit the file described on the current line of the types of files used to contain lists of files (LIST for CMS files, P-LIST for files currently being edited, CP-LIST for CP files, and TELL for further TELLs). If the current file is not one of these types or the current line is not in the correct format, a P-LIST file showing all files currently being edited will be generated and edited.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: MACRO P-EDIT

PAUSE

IMM #REINPUT PAUSE

This PF-key is used to tell P-EDIT that the user does not intend to interact with P-EDIT for a while and it can use the time to save files on disk and reclaim storage.

If P-EDIT was in INPUT mode, it will be restored when complete. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: MACRO PAUSE

PUSH/POP MASK NAME

IMM #PUSHPOP

This PF-key is used to get a new default mask name or drop the latest. If pushed once, a new default mask name will be generated. This will prevent a MASK command (with no explicit mask name) from destroying previously established masking information. If pushed twice, the latest default mask name will be cleared.

The PF-key is considered to have been pushed twice if the current default mask's value is TRUE when pushed.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

```
See: CONCEPT PARAMETRIC FILES
     MACRO   PUSHPOP
     COMMAND PUSHMASK
```

RANGE SET

```
IMM #RANGE
```

This PF-key is used to define the current line as being the first or last line of a range of lines. The next MOVE, COPY or SORT will operate on these lines if no other specification is made. The first time this PF-key is used it defines the current line as the first, the second time the last, and the third time both definitions are forgotten.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: MACRO RANGE

REPEAT

```
IMM #"
```

This PF-key will do the previous P-EDIT command again.

If P-EDIT was in INPUT mode, the effect is merely to wrap up the lines so far inserted as the effect of one INPUT command and re-enter INPUT mode. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: CONCEPT MEMORY
     COMMAND "

RETRIEVE

```
RETRIEVE
```

This PF-key is the RETRIEVE-key. If used, CP will put the last line entered into the input area. If pushed again, the line previous to that line will be placed into the input area, and so forth.

REVERSE LISP SCROLL

```
IMM #COVERT -P-SCROLL LISP
```

This PF-key moves the current line up to the last start of a LISP function definition if one can be found close enough for some text to overlap between the text displayed before and after. Otherwise, it moves the current line as much as possible to preserve some overlap. The test for the start of a LISP function is a left parenthesis in column 1 or 2.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: MACRO P-SCROLL

REVERSE SCROLL

```
IMM #COVERT -SCROLL
```

This PF-key moves the current line up as much as possible so that the text displayed still overlaps the text that was displayed.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: COMMAND SCROLL

REVERSE TELL SCROLL

```
IMM #COVERT -P-SCROLL TELL
```

This PF-key displays the previous screen of a TELL file.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: MACRO P-SCROLL

SCROLL

```
IMM #COVERT SCROLL
```

This PF-key moves the current line down as much as possible so that the text displayed still overlaps the text that was displayed.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: COMMAND SCROLL

TAB

```
TAB <tab1 <tab2 ... >>
```

This PF-key is the TAB key. CP will simulate a tab-key on a typewriter when used. To see the the current tab settings for any file, enter 'TABS' while in that file.

TELL SCROLL

```
IMM #COVERT P-SCROLL TELL
```

This PF-key displays the next screen of a TELL file.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: MACRO P-SCROLL

UNDENT

```
IMM #REINPUT CHANGE /  //
```

This PF-key is used to "unindent" the current line. "Unindenting" removes the first two double blanks.

If P-EDIT was in INPUT mode, it will be restored after unindenting the line. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

UP

```
IMM #UP
```

This PF-KEY moves the current line to the previous line in the file.

If P-EDIT was in INPUT mode, it no longer will be. The last character is a blank (not shown) so that PRESUME will recognize it as a command.

See: COMMAND UP