The Cal Computer Center Users Guide

Volume III - The 6400 Computer System

Part Three - The Time-Sharing System

Time-Sharing System Manual
May 1971 Revision

The attached version of the TSS manual completely replaces the existing documents (dated February 1971). Although significant changes have not been made to all pages, the manual has been regenerated as a whole in order to implement a new page heading format which includes the section number along with the section title in the upper right corner of each page. Where the text itself has been altered, a bar appears in the right margin. Bars appear on the following pages:

PART THREE

Ch. 1 Introduction: 1,2,4,5,8,9,10
2.2 Editor: 1,2,3,8,11,12,13
2.3 Line Collector: 1,12-15
2.4 SCOPE Simulator: 1,2,3
2.6 BASIC: (new section)
2.7 BCPL: (new section)
2.8 Printer Driver: 1,3
3.1 Files: (new section)
3.4 Capabilities and C-lists: (new section)
3.6 Event Channels: 2
4.1 File Actions: (new section)
4.4 C-list Actions: (new section)
4.6 Event Channel Actions: (new section)
Appendix A: (new)
Appendix B: (new)

PART FOUR

E. BASIC: 2-10,14,15,18,19,21
F. BCPL: (new section)

Foreword

This portion of the CAL Computer Center Users Guide is devoted to the Time-Sharing System. The Table of Contents which follows represents a tentative plan for the organization of material on the system and may be altered and/or expanded as the development of the system and the documentation progresses. Descriptions of programming languages and their processors are presented in PART FOUR of the Users Guide; therefore, information on how to use a particular language under TSS will be found in that part of the Guide.

Sections will be made available as they are completed; a date in the left-hand column of the Table of Contents indicates the existence of a section and when the latest version was released. Suggestions for improving the accuracy, clarity, and/or completeness of the documents are welcomed and should be reported to Marianne Bentley (2-1491). A suitable reward is offered for valid suggestions.

Update packages will be issued when a reasonable number of changes have been made. They will include a new Table of Contents with the date of issue at the top of the page and the new pages with bars in the left-hand margin to indicate where changes have occurred.

Preface

Use of this manual

CAL TSS is a time-sharing operating system available to users of the Computer Center. Chapter 1 contains folksy bits of information to help the novice get acquainted with CAL TSS and get a feeling for its capabilities and usefulness. Chapter 2 tells how to talk to the system via the command processor and various subsystems currently available; parts of it will be essential tc every user. Chapter 3 contains sufficiently detailed information about system concepts and structure to be of interest to a system programmer and can probably be skipped by the casual user without dire consequences. Chapter 4 gives the details of system-implemented actions which a user may invoke in code he writes. These actions may be considered as extensions to the 6400 hardware and are of interest mainly to subsystem-implementors and machine-language programmers. Chapter 5 gives details on the I/O interfaces which would allow a user to establish his own printer driver, for example.

CHAPTER 1 - WHAT CAL TSS IS AND HOW IT RUNS

1.1 INTRODUCTION TO CAL TIME-SHARING SYSTEM

CAL TSS is a large-scale, general-purpose time-sharing system written by the Computer Center staff to run on a CDC 6000 series machine with ECS. The broad design goals of the system are:

1. to support up to 256 simultaneous interactive users at teletype-compatible terminals with fast response times for simple interactions, low system overhead, and good access to various hardware facilities;

2. to provide a file system allowing many files to reside permanently in the machine and to provide a very general, powerful framework within which such files can be accessed, shared, and protected;

3. to provide a system environment in which a large number of user-oriented subsystems can be developed and run;

4. to make possible a guaranteed response time for some subset of the users of the system;

5. to utilize efficiently the hardware represented by the Computer Center's 6400B system.

CAL TSS is called "large-scale" because it is in primary control (i.e., does not run under another system) of the computer, which is a large-scale machine. It is a "time-sharing" system because a large number of users at terminals may have programs active simultaneously and may each command responses from their programs on a time-scale of a few seconds. Finally, it is called "general-purpose" because the terminal user is not restricted to some particular programming language or set of programming languages; he may, in fact, program in machine language if he so desires. In general terms, the system provides facilities for the interactive user to

1. create files, preserve them in the system, retrieve and destroy them;

2. manipulate text files with a text editor;

3. process files with a number of subsystems provided, including

a. a SCOPE simulator, giving access to all the facilities of the SCOPE system, including RUN Fortran, COMPASS, SNOBOL, etc.;
b. a BASIC processor;
c. BCPL (a low level systems programming language);

4. prepare and run his own subsystems which may interact with his teletype and other subsystems;

5. access the card reader, line printer, tape drives, and display console;

6. give access privileges for his objects selectively to other users if he so desires and obtain privileges of access to objects of other users who wish to grant it.

When the terminal load on CAL TSS is low, another system facility will process a subset of the batch jobs normally processed by the SCOPE system. Other facilities can be implemented as determined by the needs of the computing community, the programmer time available, and the capacity of the hardware.

The structure of CAL TSS and the methods of using it are extensively described in the subsequent pages of this document. Here are briefly described concepts which all users of the system will have to deal with, whether his aim is to run FORTRAN or to implement a language processing system of his own.

Figure 1. CAL T Hardware Configuration

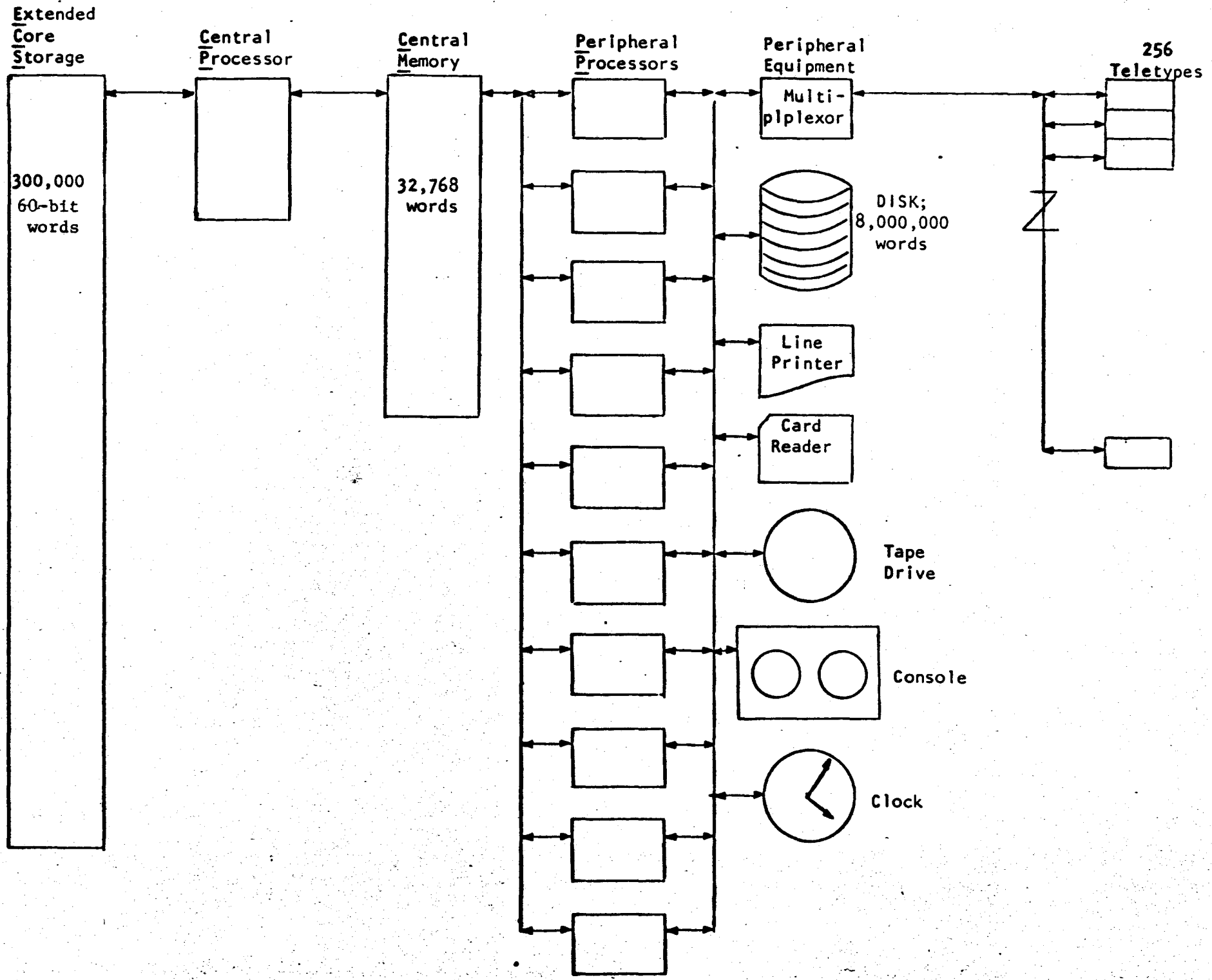
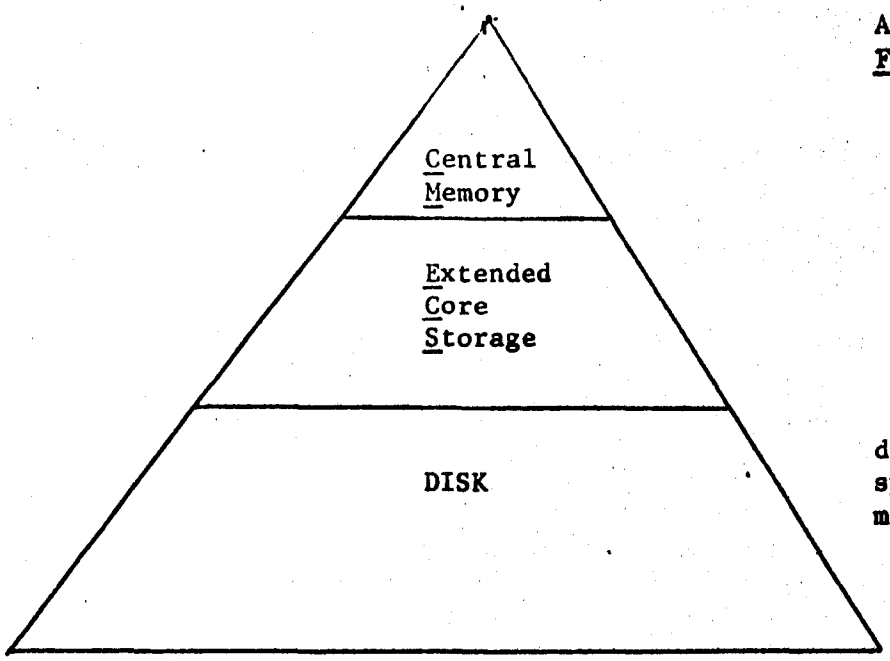


Figure 2. Storage Pyramid



Access Time For 1 word	Words Available	Cost per word
1 μs	32,768	$14
4 μs	300,000	$1
depends on system load; many ms.	8,000,000	5¢

Figure 1 indicates the hardware configuration of the current 6400B system, on which CAL TSS is run. An exact understanding of all the boxes and their interconnections is, fortunately, unnecessary, but a brief description of the memory hierarchy will make dealing with the system more understandable. Note first that a file is the basic entity for storing information in the system; code ready for execution on the central processor and text ready to be fed to a language translator are both maintained in files.

Figure 2 represents as a pyramid the different storages present in the hardware. As the figure indicates, storage at the bottom is slow and cheap and big, while storage at the top is fast and expensive and relatively small.

The disk, at the bottom of the pyramid, contains all the files which are accessible to the system without outside intervention such as mounting a tape or reading a card deck.

At the top, central memory contains the actively executing code of one process.[1] As the central processor is switched from one process to another, the code is swapped between CM and ECS. Thus, ECS must contain the code files for all processes currently active on the system.

Part of any file which is being manipulated by one of the active processes (an open file) may be in ECS or on the disk at the discretion of the process concerned. It may explicitly ask the system to maintain parts (blocks) of the file in ECS by attaching them and may dismiss blocks by detaching them. When a process accesses part of a file which is currently in ECS, the information is delivered immediately. Access to part of a file not in ECS causes the process to be blocked (stop running) until the system is able to bring the required information in from the disk. A process may, by attaching a block of a file in advance of its need to use it, improve its real-time processing speed at a cost of using more ECS.

Within this context, some of the terms with which every user will eventually become acquainted are now defined.

User profile: When someone makes arrangements to utilize CAL TSS, a body of information is recorded with the Computer Center business office which identifies him and describes his funding and access to

---

[1] The normal user at a terminal may consider all his interactions with the system to be carried out under the auspices of his own private process.

various system facilities. This information is called the user profile.

Log on/Log off: When a user attracts the attention of CAL TSS from a terminal, if his user profile and funding are in good shape, he is logged on. This is a procedure which gives him access to his own objects and such system objects and resources as his user profile allows. System resources, such as memory space, are reserved for his use at this point and may be charged against his account. When the user logs off, the resources are released and charging ceases.

File: As already noted, files are the basic entity for storing information in the system. Program code ready for execution as well as input to and output from language processors reside in files.

Directory: Directories are special objects which control access to other files, other directories, and other system objects. Directories also provide the mechanism for associating symbolic names ("print names") with the objects controlled by the directory.

Permanent disk space: An amount of space determined by the user profile is permanently reserved for a user's files. It is the only system resource tied up by a user who is not currently logged on and is charged for continuously. It is controlled by the user's permanent directory.

Temporary disk space: When a user logs on, space on the disk, as determined by his user profile, is reserved to hold the temporary files he may need while running, such as output files and compiler scratch files. This space is controlled by his temporary directory.

Process: A process may be thought of as an organizational entity within the system which ties together certain code from files and other resources necessary to "carry out a task" or "run a program". CAL TSS creates a process for each user as he logs on the system. His process looks like a 6400 central processor with somewhat less than 32K of memory; the full range of 6400 CP instructions is available to it. In addition, the user's process is able to manipulate files and certain other system-defined objects in a general way. It is delivered already equipped with some code which can, for example, communicate with the user at his teletype. The private process created for the user is given access to his permanent directory (among others), thus giving him access to his files without giving access to other users.

Fixed ECS space: Various data relevant to the state of a process are kept in ECS by the system, as are certain other objects germane to its functioning. Also, the control information for open files is kept in ECS. Because the system has no facility for keeping such information on the disk, it is kept in what is called fixed ECS space. The amount

of fixed ECS which is set aside for a given user's process is determined by his profile when he logs on.

Swapped ECS space: Files and directories which are currently in ECS at the request of a user's process are kept in what is called swapped ECS, so that the system can free ECS space if it needs to by swapping the files out to the disk. The amount of swapped ECS available to each user is also determined by his profile at log on time.

2.2 THE EDITOR

Name:	Editor
Code:	Ed 1.0
Author:	James Morris, Computer Science Department, University of California, Berkeley
Date:	November 1970
Environment:	CDC 6400: Time Sharing System

2.2.1 Purpose

The Editor subsystem provides the TSS user with facilities for constructing and editing files of coded information. A file consists of lines made up of coded characters ending with the carriage return character (generated by the RETURN key on the teletype).

2.2.2 Usage

1. Calling the Editor

The Editor is called by issuing a command of the form:

EDITOR fname

where fname is the name of the file to be created and/or edited. If fname is omitted, NULL is assumed. Whenever the file specified does not exist, an empty file is created.

The Editor responds to being called by typing its prompt character, a colon(:), and then awaiting requests from the user.

2. Editor Requests

At any given time the Editor is looking at a specific line of the file fname, called the current line. When the Editor is called, the current line is a pseudo-line which is always the top line of a file; the significance of this line will become apparent later.

The following requests may be typed to move about the file for the purpose of creating, deleting, or editing text lines. Each request is terminated either by a carriage return, or if more than one request is typed on one line, by a semi-colon. However the Editor does not actually receive the request(s) until a carriage

return has been typed. Several requests contain a "stop condition", represented by sc below, which specifies how many lines the request affects.

Requests	Meaning
I	Insert the line(s) which follow
Dsc	Delete the specified lines
T	Move to the top of the file
Msc	Move forward over the specified lines
Bsc	Move backward the specified number of lines (sc must be an integer)
Psc	Print the specified lines
C/str1/str2/sc	Replace the first occurrence of str1 by str2 in the specified lines
CG/str1/str2/sc	Replace every occurrence of str1 by str2 in the specified lines
Esc	Edit the specified lines using the line editor
R,fname,uname	Insert the contents of the file fname, uname after the current line.
W,fname,uname,sc	Locate (or create, if necessary) the file fname,uname and write the specified lines (including the current line) onto it
F,fname,uname	Finished - create the file fname from the latest edited version
Q	Finished but do not write a new file.

The stop condition, sc, may have any one of five different forms:

a number
.str where str is any string of characters not containing
/str a semi-colon.
$
absent

A number specifies the exact number of lines, including the current line, affected by the request; (the pseudo-line is also counted as a line); .str specifies all lines up to and including a line containing the character string str as a subpart; /str specifies all lines up to and including a line beginning with str (ignoring leading blanks); $ specifies all lines from the current line through the last line of the file; and the absence of sc usually (but not always) means one line.

If the last line of the file is reached before the stop condition is met, the Editor types *BOTTOM and waits at the last line of the file (except in the M command) for another request. Whenever the

Editor does not recognize a request, it types ???? and waits for a request it understands.

Insert Request: I

This request must be followed immediately by a carriage return (i.e., not a semi-colon). All characters and lines typed following it, even lines intended as requests, are inserted in the file after the current line. When a new file is being created or when information is being inserted at the beginning of a file, the current line is the pseudo-line mentioned above as being at the top of all editor files.

The end of an insertion is signaled by a carriage return at the beginning of a line; only then will editor requests be recognized.

For example, the following sequence of lines

```
EDITOR POEM
:I
JAMES JAMES
WEATHERBY GECRGE DUPREE
TOOK GREAT
CARE OF HIS MOTHER
THOUGH HE WAS THREE.
cr
M/JAMES;I
MORRISON MORRISON
```

calls the Editor (which responds with a colon), inserts five lines into the file POEM (the end of the insertion is signaled by a carriage return at the beginning of a line which appears as a blank line), then moves to the first line and inserts the line MORRISON MORRISON between the first and second lines. POEM now consists of:

```
JAMES JAMES
MORRISON MORRISON
WEATHERBY GECRGE DUPREE
TOOK GREAT
CARE OF HIS MOTHER
THOUGH HE WAS THREE.
```

Additional lines can be added tc the end of the file by moving to the bottom of the file and inserting.

```
M$;I
JAMES JAMES
MORRISON MORRISON
WEATHERBY GECRGE DUPREE
```

```
ONE RAINY MORNING
SAID IO THIS MOTHER,
'MOTHER,' HE SAID, SAID HE:
'YOU MUST NEVER GO DOWN TO THE END OF THE TOWN,
IF YOU DON'T GO DOWN WITH ME.'
cr
```

The last line of text inserted becomes the new current line. The file POEM now contains:

```
JAMES JAMES
MORRISON MORRISON
WEATHERBY GECRGE DUPREE
TOOK GREAT
CARE OF HIS MOTHER
THOUGH HE WAS THREE.
JAMES JAMES
MORRISON MORRISON
WEATHERBY GECRGE DUPREE
ONE RAINY MORNING
SAID TO HIS MOTHER,
'MOTHER,' HE SAID, SAID HE:
'YOU MUST NEVER GO DOWN TO THE END OF THE TOWN,
IF YOU DON'T GO DOWN WITH ME.'
```

Delete Request: Dsc

The delete request erases lines in the file beginning with the current line and ending with the first line satisfying the stop condition. Specifically, if the current line were the eighth line of POEM, i.e., the second MORRISON MORRISON, then

```
D.RAIN or D/ONE
```

would delete three lines, leaving:

```
JAMES JAMES
MORRISON MORRISON
WEATHERBY GECRGE DUPREE
TOOK GREAT
CARE OF HIS MOTHER,
THOUGH HE WAS THREE.
JAMES JAMES
SAID TO HIS MOTHER:
'MOTHER,' HE SAID, SAID HE:
'YOU MUST NEVER GO DOWN TO THE END OF THE TOWN,
IF YOU DON'T GO DOWN WITH ME.'
```

Deleted lines are replaced temporarily by a line which prints as *DELETED; it disappears however, as soon as the next move in the file is made. One or more lines can be replaced by combining the delete and insert requests. Specifically, if the current line in the file POEM is: THOUGH HE WAS THREE, the requests:

```
D;I
THOUGH HE WAS ONLY THREE.
cr
```

produces the following version of the poem:

```
JAMES JAMES
MORRISON MORRISON
WEATHERBY GEORGE DUPREE
TOOK GREAT
CARE OF HIS MOTHER
THOUGH HE WAS ONLY THREE.
JAMES JAMES
SAID TO THIS MOTHER,
'MOTHER,' HE SAID, SAID HE:
'YOU MUST NEVER GO DOWN TO THE END OF THE TOWN,
IF YOU DON'T GO DOWN WITH ME.'
```

which, incidentally, is the true version.

Move to Top Request: T

The "top" requests asks the Editor to move to the top of the file, and the empty line becomes the new current line. Thus to insert text at the beginning of the file, one might type:

```
T;I
TITLE:
'DISOBEDIENCE
BY A.A. MILNE
```

Now the file would appear as follows:

```
TITLE:
DISOBEDIENCE
BY A.A. MILNE

JAMES JAMES
MORRISON MORRISON
WEATHERBY GEORGE DUPREE
```

```
TOOK GREAT
CARE OF HIS MOTHER
THOUGH HE WAS ONLY THREE.
JAMES JAMES
SAID TO HIS MOTHER
'MOTHER,' HE SAID, SAID HE:
'YOU MUST NEVER GO DOWN TO THE END OF THE TOWN,
IF YOU DON'T GO DOWN WITH ME.'
```

The combination of requests, T;D*sc* cr, can be used to delete lines at the beginning of a file. For instance,

T;D.LE:

or T;D/TI

or T;D2

will delete the first line of the file POEM leaving the lines from DISOBEDIENCE on down to the end.

Note that to delete the *first* line of the file using a numerical stop condition, the number 2 was required rather than 1 or simply D. Recall that T positions the editor at the pseudo-line, rather than the first line of text, so that although it is not possible to delete the empty line, it must be included in the count for the stop condition. Indeed this consideration applies to numerical stop conditions for all requests issued immediately following the top request.

Move Request: M*sc*

The move request changes the current line to the first line which satisfies the stop condition. For instance, in the latest version of POEM, the requests

T;M9

or M/TOOK

will both position the Editor at the line

TOOK GREAT

regardless of the old current line position. Whenever the last line in the file is reached before the particular stop condition is met, the Editor continues searching at the top of the file (i.e., it wraps around). If the condition is never met, it types *NOT FOUND and returns to the old current line.

Since M$ sends the Editor to the last line of the file,

```
.M$;I
new line(s)
cr
```

is the easiest way to append lines to the file. Note that the example on page 4 uses this method.

Backward Move Request: Bsc

The backward move request moves the current line position backward in the file the number of lines specified by the stop condtion. A number is the only stop condition allowed with the B request. Thus

```
B
```

or

```
B1
```

moves the current line position back one line; B7 will move it back 7 lines. However, an attempt to back up beyond the first line of text causes the Editor to complain:

```
* BACK TOO FAR
```

and remain at the current line position.

Print Request: Psc

The print request asks the Editor to print out all lines up to and including either the line satisfying the stop condition or the last line, whichever the Editor encounters first. The last line printed becomes the new current line. Specifically, P or P1 prints the current line but causes no change in line position. P2, however, is equivalent to P;M;P. Thus in the file POEM as it stands so far:

```
T;P2
```

evokes

```
DISOBEDIENCE
```

(Recall that the pseudo-line is included in the count.)

```
P3
```

will then cause

```
BY A.A. MILNE

JAMES JAMES
```

to be printed. The blank line, of course, counts as a line.

```
P/MORRIS
```

prints the first and second lines of the poem, then

```
P$
```

prints the second line through the end.

If the end of the file is reached before the stop condition is satisfied, the Editor types out

```
*BOTTOM
```

and remains there.

If the Editor has been requested to print out far more than was actually desired (i.e., is in Sorcerer's Apprentice mode), it can be stopped by typing CRTL-SHIFT-P. The Editor responds with a colon and then awaits the next request.

Change Request: C/str1/str2/sc

This request enables the user to alter a line without retyping it, as was required by the Dsc;I technique. The first occurrence (if it exists) of the character string str1 is replaced by str2 in the current line and all succeeding lines through the line satisfying the stop condition. For example, the sample poem can be changed further by such requests as:

```
T;M/JAMES;C/MES/NE/
```

which alters the first line of the poem to:

```
JANE JAMES
```

Then, however, the subtlety of the Editor's ways is demonstrated when:

```
C/MES /NE/
```

causes the Editor to respond

```
*NO CHANGE*
```

The idea was to change the second occurrence of JAMES in the first line to JANE. Since it occurred at the end of the line however, it was not followed by a blank. Therefore, the Editor was unable to find the string to be changed and reported its failure with the *NO CHANGE message.

However, C/MES/NE/ produces the following version of the poem:

```
DISOBEDIENCE
BY A.A. MILNE

JANE JANE
MORRISON MORRISON
WEATHERBY GEORGE DUPREE
TOOK GREAT
CARE OF HIS MOTHER
THOUGH HE WAS ONLY THREE.
JAMES JAMES
SAID TO HIS MOTHER,
'MOTHER,' HE SAID, SAID HE:
'YOU MUST NEVER GO DOWN TO THE END OF THE TOWN,
'IF YOU DON'T GO DOWN WITH ME'.
```

The following series of requests make the poem look more like the original by indenting all but the second to last line:

```
T;M/DIS;C//     /12
M$;C//     /
```

Any character that is not a letter may be used instead of slash to delimit the string. For instance

```
C..     .12
```

would also indent 12 lines.

Global Change Request: CG/str1/str2/sc

This command is identical to the change request except that all occurrences of str1 in the current line are replaced by str2. For example,

```
T;CG/HE/SHE/$
```

produces the following version of POEM by changing all occurrences of the word HE to the word SHE:

```
DISOBEDIENCE
BY A.A. MILNE

JANE JANE
MORRISON MORRISON
WEATHERBY GEORGE DUPREE
TOOK GREAT
CARE OF HIS FATHER
THOUGH SHE  WAS ONLY THREE.
JAMES JAMES
SAID TO HIS FATHER:
'FATHER,' SHE SAID, SAID SHE,
'YOU MUST NEVER GO DOWN TO THE END OF THE TOWN,
IF YOU DON'T GO DOWN WITH ME.'
```

On the other hand, T;C/HE/SHE/$ would have allowed the second HE on line 12 to escape unchanged.

Edit Request: Esc

The edit request offers an alternate method for changing the text of the current line. It causes the current line to be treated as if it had just been typed in, and the user is expected to enter line editor requests (see the section on the Line Editor) to specify a new one which will replace it. As soon as the edited line is completed and has been accepted, the Editor no longer accepts Line Editor requests unless the edit request is retyped. For example, if the current line is:

```
THOUGH SHE WAS ONLY THREE.
```

the word "only" can be deleted by typing:

```
C/ONLY//
```

The same edit can be done by typing:

E
CTRL-F S CTRL-F S CTRL-X CTRL-SHIFT-0

which copies the line through the second occurrance of the letter S, skips one word and then concatenates the rest of the line, prints it, and causes the new line to be accepted and substituted for the old line.

To then as an afterthought use the line editor to change the word SHE to HE, the editor request must be entered again followed on the next line by the appropriate line editing requests.

An attempt to use the Line Editor when positioned at the empty line, e.g.,T;E evokes PSEUDO LINE SKIPPED from the Editor. The Editor must be requested specifically to look at the line to be edited, before the edit can be entered.

Read File Request: R,fname,uname

This command inserts the contents of the file fname,uname [1] immediately following the current line in the file. The last line inserted becomes the current line. If, for instance, the second verse of the poem in the sample were on the file SECOND, the sequence of requests:

M$;R,SECOND;T;P$

would produce the first two verses of the poem.

Write File Request: W,fname,uname,sc

This command creates the file fname,uname [1] if it does not already exist, and writes onto it the contents of the current file from the current line through the line satisfying the stop condition. If the stop condition is omitted, it is assumed to be $ instead of 1. For instance, after moving the second verse of the sample onto the current file, the poem could be edited into its original form and written onto the file KEEP using the following requests:

T;CG/SHE/HE/$;T;W,KEEP,,$

---

[1] uname is the name of a temporary directory to which the user has access.

Finished Request: F,fname,uname [1]

This request tells the Editor that the editing is completed and asks it to return control to the Command Processor. Before doing so, it will leave the most recent version of the current file in ECS under the name fname, or if no name is given in the F request, under the fname specified in the command which called the Editor initially. In the case where the Editor was called to edit an already existing file and the editing session is terminated by an F request with a different fname, both the original file and the edited version will be kept.

Quit Request: Q

If, after editing a file, the user decides it was all a mistake, the Q request can be used instead of F. The Editor responds by leaving the original file in ECS unedited and not generating a new file.

2.2.3 Restrictions

The two change requests restrict the number of characters in str2 to 50 characters.

The Editor permits lines to consist of up to 160 characters but it is suggested that they be restricted to a total of 80 characters to assure compatability with other processors, such as the Command Processor, the Line Editor, and the teletype interface.

2.2.4 Notes

The Editor works by copying files in a purely sequential fashion; at any point one file is being read and a new one is being created with those modifications called for by the various requests.

A T (top) command causes the Editor to copy the remainder of the input file to the output file, to make the output file the new input file, and to start creating a new output file. An M command which moves past the last line of the file and returns to the top of the file, also causes this activity. The two scratch files used for this see-saw process are given the names 1fname and 2fname, where fname is the name of the file declared when the editor is called. In general, the process is as follows:

If the user calls the Editor with:

```
EDITOR   BLATZ
```

then the Editor

1) starts reading BLATZ (if it already exists) and writing 1BLATZ
2) then reads 1BLATZ and writes 2BLATZ
3) then reads 2BLATZ and writes 1BLATZ, etc.

and at the end, assuming an F command,

4) renames the most recent file written (1BLATZ or 2BLATZ) with the original name BLATZ and deletes the other two files.

Thus, in the event of a minor disaster (for example, the user deleted half the file inadvertently), a fairly recent version of the file may be found under one of these names in the user's temporary directory.

2.3 THE LINE COLLECTOR

Title:	Line Collector
Code:	LC1.0
Author:	Jim Gray, Computer Science Department, University of California, Berkeley
Date:	November 1970
Environment:	CDC 6400: Time Sharing System

2.3.1 Purpose

The line collector constructs a line from the TTY using the previously typed line as a template. It maintains two lines simultaneously, an old one and a new one. The old line is the last line received by the Teletype and is local to the virtual teletype buffer; it may possibly be empty. A new line is constructed from the old one using the characters typed in from the Teletype. To visualize the process of constructing each new line, imagine two cursors or pointers, one called OLD which runs over the old line and one called NEW which is positioned on the new line as it is created. Normally when a character is entered from the TTY, it is appended to the new line and both cursors advance one place. If certain non-graphic characters, comprising requests to the line collector (see Figure 1) are entered, the cursors can be manipulated so that, for example, characters are COPIED from the old line to the new one, or parts of the old line are SKIPped, or the cursors BACKUP over the undesired characters.

One application for the line collector would be in conjunction with an on-line compiler which performs a simple syntax check of each line as it is entered. If the line is bad, it outputs a diagnostic, rejects the line, and calls on the line collector. The user edits the old line which still resides in the virtual buffer and resubmits it to the compiler.

2.3.2 Usage

The line collector accepts a number of requests for actions to be performed on the old and/or new lines. Most of these involve moving the cursors forward or backward to the desired locations. The actions resulting from the requests are given below. In general, the actions can be separated into five categories: copy, back up, skip, accept, and other. In each of the first three categories, there are six requests for specifying the various distances in the old and new lines the cursors are to move. In the descriptions which follow, if the first key(s) specified is (are) CTRL-, or CTRL-SHIFT-, the next key must be pressed <u>while</u> the first key(s) is (are) still depressed. "word" is defined as a sequence of one or more alphanumeric characters delimited by non-alphanumerics; when looking for the beginning of a

word, the cursor disregards all non-alphanumerics until it encounters one or more consecutive alphanumerics. For instance, in the line

'MOTHER,' HE SAID, SAID HE,

the beginning of the first word is the letter M, not the graphic quote; and the end of the first word is R, not the graphic comma. "Next character entered" refers to the first occurrence in the line of the next character typed following the request. If at any time an edit request cannot be fulfilled, the line collector responds with a bell. In the examples, the cursor is represented by a vertical bar below the line.

Copy Requests

1. Copy one character: CTRL-A

The next character in the old line is appended to the new line, and the character is printed. For example, if before the request is issued, the old and new lines appeared as

```
Old:      THOUGH HE WAS ONLY THREE.
          |
New:
          |
```

the request would produce:

```
Old:      THOUGH HE WAS ONLY THREE.
          |
New:      T
          |
```

2. Copy one word: CTRL-S

The characters in the old line up to the next word boundary are appended to the new line, and are printed. For example, if the old and new lines appeared as they were left in the previous example, the request would produce:

```
Old:      THOUGH HE WAS ONLY THREE.
               |
New:      THOUGH
               |
```

If issued again, it would produce:

```
Old:    THOUGH HE WAS ONLY THREE.
                  |
New:    THOUGH HE
                  |
```

3. Copy up to next character entered: CTRL-D

Characters in the old line up to but not including the next character entered are appended to the new line and printed. Given the position of the cursors at the end of the previous example, CTRL-D W would produce

```
Old:    THOUGH HE WAS ONLY THREE.
                  |
New:    THOUGH HE
                  |
```

A second such request of the form CTRL-D O would produce:

```
Old:    THOUGH HE WAS ONLY THREE.
                     |
New:    THOUGH HE WAS
                     |
```

4. Copy up to and including the next character: CTRL-F

The characters in the old line up to and including the first occurrence in the remainder of the line of the next character typed are appended to the new line and printed. For example, using the lines in the last example, CTRL-F N produces:

```
Old:    THOUGH HE WAS ONLY THREE.
                        |
New:    THOUGH HE WAS ON
                        |
```

5. Copy to tab: CTRL-G

The characters in the old line up to the next tab setting (see Other Requests below) are appended to the new line and printed. Assuming there is a tab setting at print position 20, and the cursors are positioned as they were left from the previous example, CTRL-G produces:

```
Old:    THOUGH HE WAS ONLY THREE.
                           |
New:    THOUGH HE WAS ONLY
                           |
```

6. Copy remainder of old line: CTRL-H

The remaining characters in the old line are appended to the new line and printed. CTRL-H has the following effect on the line in the previous example:

```
Old:     THOUGH HE WAS ONLY THREE.
                                  |
New:     THOUGH HE WAS ONLY THREE.
                                  |
```

Backup Requests

1. Backup one character: CTRL-Q

One character is erased from the new line by backing up the cursor in both lines one place. ← is printed in the next forward position in the printed line. This request would have the following effect on the lines in the previous example:

```
Old:     THOUGH HE WAS ONLY THREE.
                                  |
New:     THOUGH HE WAS ONLY THREE.←
                                  |
```

and the actual image of the new line is:

```
         THOUGH HE WAS ONLY THREE
                                  |
```

2. Backup one word: CTRL-W

All characters up to the last non-alphanumeric character are erased by backing up the cursor in both lines. \ is printed in the next (forward) position in the printed line. This request would have the following effect on the lines in the previous example:

```
Old:           THOUGH HE WAS ONLY THREE.
                                       |
New (printed):THOUGH HE WAS ONLY THREE.←\
                                       |
(actual):      THOUGH HE WAS ONLY
                                       |
```

3. Backup to next character entered: CTRL-E

All characters in the new line up to but not including the first occurrence in the backward direction of the next character entered

following the request are erased by backing up the cursor in both lines. \ is printed in the next forward position. A request such as CTRL-E N would have the following effect on the previous example:

```
Old:            THOUGH HE WAS ONLY THREE.
                                 |
New (printed):THOUGH HE WAS ONLY THREE. ←\\
                                 |
  (actual):     THOUGH HE WAS ON
                                 |
```

4. Backup through next character entered: CTRL-R

All characters in the new line up to and including the first occurrence in the backward direction of the next character entered after the request are erased by backing up the cursor in both lines. \ is printed in the next forward position. A request such as CTRL-R W would have the following effect on the previous example:

```
Old:            THOUGH HE WAS ONLY THREE.
                           |
New (printed):THOUGH HE WAS ONLY THREE.←\\\
                           |
  (actual):     THOUGH HE
                           |
```

5. Backup to tab: CTRL-T

All characters in the new line up to and including the first tab setting encountered in the backward direction are erased by backing up the cursor in both lines. \ is printed in the next forward position in the printed line. This request would have the following effect on the lines in the previous example, if there were a tab setting in position 10:

```
Old:            THOUGH HE WAS ONLY THREE.
                           |
New (printed):THOUGH HE WAS ONLY THREE. ←\\\
                           |
  (actual):     THOUGH HE
                           |
```

6. Backup to edge: CTRL-Y

The new line is erased completely and may be started anew. ↑ is printed in the next print position, the printer paper is advanced

and the carriage is returned. Thus, to change the line in the previous example to read ALTHOUGH HE WAS etc., type: CTRL-Y ALTHOUGH HE WAS ONLY THREE. After the back up request, the line would appear as:

```
Old:      THOUGH HE WAS ONLY THREE.
          |
New:      THOUGH HE WAS ONLY THREE.←\\\↑
          |
```

Then typing the first word of the new line would produce

```
Old:      THOUGH HE WAS ONLY THREE.
                |
New:      ALTHOUGH
                |
```

Note that since the first word has more characters than it did in the old line, the cursor in the cld line has run into the next word. It would be handy to use the "copy to edge" request here, but to do so would make nonsense of the line, i.e., ALTHOUGHE WAS ONLY THREE. Instead the remainder of the new line must be typed in, producing:

```
Old:      THOUGH HE WAS ONLY THREE.
                                   |
New:      ALTHOUGH HE WAS ONLY THREE.
                                   |
```

It will be shown later how the "insert request" can be used to avoid this inconvenience.

Skip Requests

1. Skip one character: CTRL-Z

One character in the old line is skipped by moving the cursor in the old line ahead one character. $ is printed on the teletype. Assuming that the old line is:

```
          'MOTHER,' HE SAID, SAID HE,
         |
```

the request CTRL-Z produces

```
Old:      'MOTHER,' HE SAID, SAID HE,
          |
New (printed): $
          |
```

```
   (actual):
                |
```

2. Skip one word: CTRL-X

One word in the old line is skipped by moving the cursor in the old line ahead to the first non-alphanumeric character. $ is printed for each character skipped. This request would have the following effect on the lines in the last example:

```
Old:            'MOTHER,' HE SAID, SAID HE,
                        |
New (printed):$$$$$$$
                        |
    (actual):
              |
```

3. Skip to next character entered: CTRL-C

All characters in the old line are skipped up to the next character entered after this request by moving the cursor in the old line up to that character. $ is printed in place of characters skipped. CTRL-C H has the following effect on the lines in the previous example:

```
Old:            'MOTHER,' HE SAID, SAID HE.
                          |
New (printed):$$$$$$$$$
                          |
    (actual):
              |
```

If the cursors were positioned at the beginning of the lines, a new version of the line could be obtained in the following manner (the line will be shown after each request to illustrate the action of the cursors):

First type: CTRL-A (copy one character)

```
Old             'MOTHER,' HE SAID, SAID HE.
                |
New:            '
                |
```

Then type,
SIST CTRL-D H (enter SIST instead of MOTH and copy up to the beginning of the word HE)

```
Old:            'MOTHER,' HE SAID, SAID HE.
                          |
```

```
New:          'SISTER,'
                       |
```

Then,

I CTRL-X (enter I instead of H and skip to the end of the word; one could also say I CTRL-C (blank) to do the same thing)

```
Old:          'MOTHER,' HE SAID, SAID HE,
                        |
New:          'SISTER,' I$
                         |
```

Then CTRL-D H (again, to copy up to the beginning of HE, this time at the end of the line)

```
Old:          'MOTHER,' HE SAID, SAID HE,
                                        |
New:          'SISTER,' I SAID, SAID
                                       |
```

and finally, I, to finish the line

```
Old:          'MOTHER,' HE SAID, SAID HE,
                                        |
New (printed):'SISTER,' I$ SAID, SAID I,
  (actual):   'SISTER,' I SAID, SAID I,
                                       |
```

Note that the cursor in the old line has not yet reached the end of the line; this is irrelevant as long as a copy is not requested next.

4. Skip through next character entered: CTRL-V

All characters in the old line through the character entered following the request are skipped by moving the cursor in the old line to the character following the next occurrence of that character in the line. $ is printed for each character skipped. For example, if the cursors were positioned as in the lines:

```
Old:     'MOTHER,' HE SAID, SAID HE,
                  |
New:     'MOTHER,
                  |
```

(perhaps as the result of a copy through the next character entered where that character was a comma), the request CTRL-V D,

followed by a skip one character (CTRL-Z) and by copy to edge request would produce:

```
Old:          'MOTHER,' HE SAID, SAID HE,
                                       |
New
(printed):    'MOTHER,' $$$$$$$$ SAID HE,
(actual):     'MOTHER,' SAID HE,
                        |
```

5. Skip to tab: CTRL-B

All characters in the old line up to the next tab setting are skipped by moving the cursor in the old line to the next tab position. $ is printed for each character skipped.

6. Skip to end of line: CTRL-N

The remainder of the old line is skipped by moving both cursors to the position at the end of the old line. $ is printed for each character skipped. For example, if a new line had been constructed as follows:

```
Old:     'MOTHER,' HE SAID, SAID HE.
                          |
New:     'SISTER,' HE SAID,
                          |
```

A CTRL-N request would move the old cursor to the position at the end of the old line, leaving the new line as is.

Accept Requests

There are four requests which cause the new line to be accepted and thereby mark the end of the requests that can be made for that line. The first two cause the line to be accepted as is; the second two cause the remainder of the old line to be concatenated onto the new line before it is accepted.

1. Accept Request: cr

Depressing the carriage return key enters the accept request. The current new line is accepted as is, terminating the construction of that new line.

Special Accept: CTRL-U (not implemented)

This request is identical to the accept request except that the line collector notifies the calling routine that this line is special.

Concatenate and Accept: CTRL-P

This request appends whatever is left following the cursor in the old line to the new line and then causes it to be accepted. For instance, if only the beginning of a line were being altered as in:

```
Old:    'MOTHER,' HE SAID, SAID HE.
                 |
New:    'SISTER,'
                  |
```

a CTRL-P request would cause the new line to be accepted as: 'SISTER,' HE SAID, SAID HE.

Concatenate, Print and Accept: CTRL-O

This request is identical to CTRL-P except that the new line is printed out in its final form. It is handy when quite a few changes have been made to the old line when creating the new line.

Other Requests

There are six other requests that can be made of the line collector. They provide facilities for inserting text in a line, setting and releasing tabs, tabbing, printing the current state in both lines, and concatenating and reediting without having the line accepted.

Insert Request: CLTR-L

When entered an odd number of times since the beginning of the new line, the cursor in the old line is not moved on Backup operations or when characters are entered, thereby allowing the insertion of characters into a line. < is printed in response to odd numbered entries of the request. Even numbered entries return the old cursor to its normal action and cause > to be printed. Suppose, for example, the line THOUGH HE WAS THREE is to be changed to: ALTHOUGH HE WAS ONLY THREE. The following series of requests could be used:

CTRL-L AL CTRL-L inserts the characters AL at the beginning of the line:

```
Old:          THOUGH HE WAS THREE.
              |
New
(printed):    <AL>
(actual):     AL
               |
```

Then CTRL-F S CTRL-L ONLY CTRL-O copies the old line through the letter "S", inserts the word "only", and concatenates the remainder of the old line, causes it to be accepted, and prints the line as it was accepted. The output on the teletype would appear as:

```
<AL>THOUGH HE WAS < ONLY  THREE.
ALTHOUGH HE WAS ONLY THREE.
```

Tab Set/Release Request: CTRL-K

This request sets (releases) a tab stop at the current position of the cursor in the new line when entered an odd (even) number of times.

Tab Request: CTRL-I

This request inserts blanks up to the next tab stop (both cursors advance). Blanks are inserted in the printed line. For instance, if the user wanted to indent a number of lines five spaces, and the old line started in print position one, he could set a tab (see above) in print position 5 and start the new line with request CTRL-I. Subsequent indented lines could be begun with either CTRL-I or CTRL-C (skip to next character entered) followed by the first letter of the previous line.

Type State Request: CTRL-SHIFT-K

This request allows the user to see the current state of the editing process; the printer paper is advanced to a fresh line, the carriage spaces to the current position of the new cursor, prints a copy of the remainder of the old line, and on the following line prints a copy of the new line up to the current position of the new cursor. For example, if CTRL-SHIFT-K has been typed after CTRL-F S ("copy through the character S") in the last example, the printed output would have been as follows:

```
<AL>THOUGH HE WAS
                THREE.
ALTHOUGH HE WAS
```

indicating that, in the old line, the cursor was positioned before the word THREE and then printing the new line as it had been constructed so far.

Concatenate and Re-edit Request: CTRL-SHIFT-L

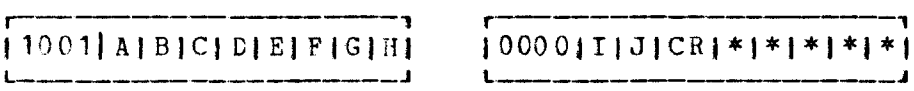
This request combines some aspects of two of the requests described above by appending the remainder of the old line onto the new line, but instead of causing it to be accepted, it makes this line the old line with the cursor positioned at the beginning. The printer paper is positioned at the beginning of a fresh line. Thus the "new line" can be edited further before being accepted.

2.3.3 TSS Text Standard

The System Standard Text (Systext) is the standard method of storing coded information for the Time Sharing System. Information in Systext format exists in a file (a semi-infinite array of 60-bit words) and is terminated by an end-of-information word. A Systext file is composed of lines, which contain character coded information and segments called sloppy segments which contain no information.

Systext Lines

A line is a sequence of 7 bit ASCII characters terminated by the control character CR (= 155B). There is no limit to the length of a line, and they may be split across file block boundaries. Each line is packed left-justified into successive 60-bit words, 8 characters (56 bits) per word. The first 4 bits of each word serve to signal the beginning of a line: for the first word of a line these leading bits are 1001; for all other words in a line they are 0000. Consider the lines ABCDEFGHIJ CR which would be stored in Systext as:



Characters which follow the appearance of CR in a word are ignored.

Multiple blanks in a line are compressed by inserting a count of the number of blanks rather than the blanks themselves. The ASCII character ESC (= 173B) is reserved for this purpose. Whenever ESC occurs in the Systext file, the character following it is interpreted

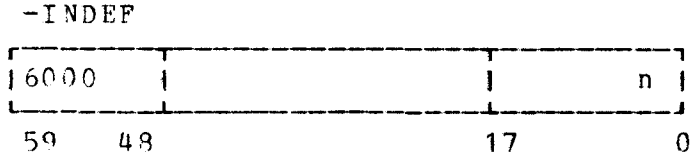
as a blank count, 'n' ($0 \leq n < 128$). On output these two characters are replaced by n blank characters.

Character Representation

The internal ASCII code used in System Standard Text is the external ASCII + 140B (mod 200B). The conversion is performed by the system I/O routines. This scheme maps blank onto 0, 0 onto 20B and A onto 41B. See Appendix A. Table 1. Non-graphic characters, however, are not allowed to occur in System Standard Text. (Carriage return and ESC in the context described above are the only exceptions.) Therefore, the character % has been reserved as a special prefix for representing non-graphic characters; if the graphic following a % maps onto a control character under the mapping: internal ASCII + 100B (mod 200B), the pair is interpreted as that control character (see Appendix A Table 2). Otherwise the % leaves its successor unchanged. So %% represented % and %M represented CR.

Sloppy Segments

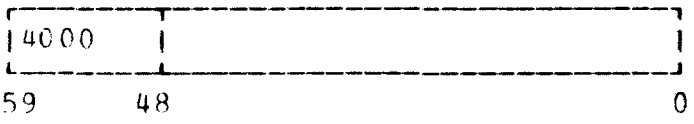
A sloppy segment in the Systext file is a group of n words ($0 < n < 2^{18}$) that are to be ignored. The first and last word of such a segment are of the form:



where n is the count of words in the segment. The system ignores the middle 30 bits of this header word and the successding n-1 words. A sloppy segment may not occur within a line and cannot be split across file block boundaries.

End-of-information

The end of Systext is signaled by an end-of-information (EOI) word of the form:



The low order 48 bits of the word are ignored.

2.3.4 Line Collector Actions

Calling the Line Collector (LC:ASCII)

The line buffer associated with a teletype is a sequence of words containing Systext, headed by a word specifying a pseudo-character count. The Systext may either contain compressed blanks or not, depending on how the buffer is filled. Four actions are available for manipulating the buffer: input a line, output a line, output a character, and edit a line. The input parameters are:

IP1 D: action specifier
IP2 BD: line buffer holding ≤85 characters)

where the action specifier may be

IP1=0	no operation
IP2=1	Input a line. (IP2 is ignored). A line is read in frcm the teletype and is then returned to the user via the block data return authorization, i.e., the block of words constituting the line buffer is returned in the parameter RDAT BD: line buffer. Blank characters are not compressed.
IP1=2	Output a line. The contents of the line buffer specified by IP2 is cutput to a teletype up to a carriage return character, or when the pseudo character count in the header word is exhausted. If the line contains compressed blanks, the pair: ESCn is treated as 2 pseudo-characters.
IP1=3	Output a character. The first word of IP2, which is assumed to contain a single right-adjusted Systext character, is sent to a teletype.
IP1=4	Edit a line. The line passed in IP2 becomes the old line. (Blank compression is not allowed.) A line is then read from the teletype and returned to the user as in IP1=1.

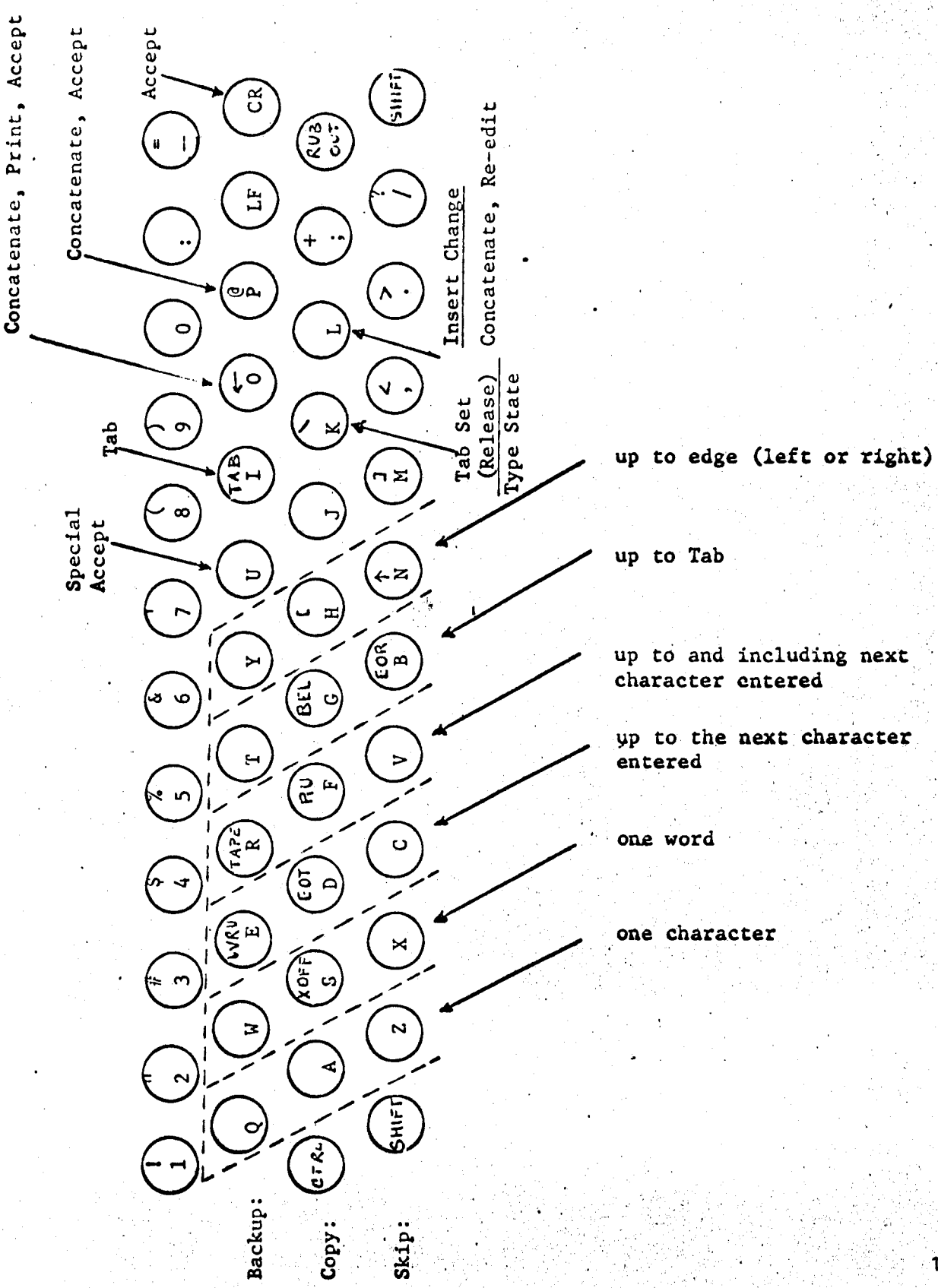
NOTES

1. The line collector limits the size of a line to 85 characters. Should a line be constructed having more than 85 characters using an insertion, all characters following the end of the insertion are dropped. If a line exceeds 80 characters, the Concatenate Print and Accept command will not print the line although the other functions are still performed.

2. Whenever a request produces the effect of another request of a more specific nature, the line collector responds as if the more specific request had been entered. For instance, if the cursors are positioned at the end of the first word of a line and the request to "back up one word" is given, the line collector responds as if the request "back up to edge" had been given. Similarly, if the cursors are positioned at the second letter of a word (which is not the first word of a line) a "back up one word" request causes a ← (for "back up one character") to be printed rather than \ .

3. A special character in the middle of a word, e.g., DON'T or P.OPS, is interpreted by the line collector as a word boundary. For example, if the cursors are positioned at the beginning of the word DON'T, a "copy one word" produces DON instead of the full word. A second "copy one word" copies 'T, but then a "back up one word" interprets ' as the word boundary and stops at the T, printing ← (see note 2).

Figure 1. (33/35) Teletype Keyboard and Control Characters



2.4 THE SCOPE SIMULATOR

Title:	SCOPE Simulator
Code:	SC1.1
Author:	Karl Malbrain, Computer Center, University of California, Berkeley
Date:	February 1971
Environment:	Machine: CDC 6000 Series Operating System: CAL DISK TSS Coding Language: COMPASS

2.4.1 Purpose

The SCOPE Simulator provides an operating environment for programs written for Cal's 6400 batch system (SCOPE 3.0 or CALIDOSCOPE) and permits interactive control by the user over the construction and execution of such programs. It also provides a batch run facility for background execution of small jobs.

2.4.2 Method

The Simulator runs in two subprocesses in the user's process. The lower subprocess contains the simulator itself while the upper one contains the "user's subprocess", i.e., the environment for the SCOPE program (e.g., registers, core, RA+1 checking, etc.).

Local files for the program as well as interfaces into the user's permanent files and the system library files are generated.

The Simulator indicates that it is ready to accept requests from the user by causing a "greater than" (>) symbol to be printed on the teletype. The appearance of an "up arrow" (↑) signals that the program itself is awaiting a control request.

2.4.3 Usage

The SCOPE Simulator is called by issuing the following request to the Command Processor:

SCOPE f1

where f1 is a field length. In general, the field length may range from 1000 to 50000, with 14000 assumed if the parameter is omitted. SCOPE will type out the time and date and then await the first command after typing >. At this point any legal CALIDOSCOPE control request can be typed in immediately following the > symbol. For example,

```
>RFL,50000
>RUN.
>LGO.
```

would increase the field length to 50000 (octal), compile a FORTRAN program, and then read and execute it.

SCOPE Simulator requests may also be entered after the >.

The following are legal SCOPE Simulator requests:

SYSTEXT File Declaration: TEXT,fname

The file fname is declared as a new SYSTEXT file and information written on it by the running program will be translated to SYSTEXT as it is written. For example,

```
>TEXT,OUTPUT
```

would declare the file name OUTPUT to be a Systext (See Section 2.3.3) file (although it already is by default). Files that already contain information in Systext (generated by the Editor, for example) need not be declared. The proper conversion will be performed on reads (or writes). This request will not convert into a systext file a file which already exists in some other mode.

Request File Execution: FILE,fname

The file fname contains a list of requests which will be interpreted one by one as separate requests. An imbedded FILE request causes a transfer to the new file. All messages that would normally be typed out if the requests were entered individually are suppressed, except error messages. When an error occurs, the file is advanced past the next EXIT request or to the end-of-file, whichever occurs first. > is typed if an EXIT request is reached normally, or if the end-of-file is reached.

Control Message Printing: MSG,OFF or MSG,ON

Normally, messages from the program are printed on the teletype. These may be suppressed by:

```
>MSG,OFF
```

or restarted by:

```
>MSG,ON
```

All messages are written on the System log, however, regardless of whether they appear on the teletype.

Handling Remote Files: GET,fname
PUT,fname

The first time the Simulator references any file, it attempts to obtain it from the user's temporary directory. Files in other directories may be obtained via the GET request. The specified file name must never have been used by the program before. For example,

```
>GET,LGO
```

requests that the file named LGO be obtained from the Command Processor. The file may be closed and forgotten explicitly by the PUT request:

```
>PUT,LGO
```

Files which are not disposed of explicitly are returned when the user finishes with the Simulator during FIN request processing (see below).

Step Mode: STEP

The STEP request allows the user to trace the 'RA+1' calls which the program makes on the Simulator. This mode of tracing each request is entered via:

```
>STEP
```

The Simulator prints each request in octal before it is performed, then stops and waits for verification after printing a greater than sign (>). There are four responses that can be made, each consisting of a single letter:

B	means call the BEAD ghost Debugger. Upon return, another > is printed.
S	means perform the request.
E	means ignore the request and perform an END request instead.
G	means leave STEP mode and then do the request.

Any other letter is equivalent to S.

Loading Requests

The simulator also provides the GPSL functions of loading. These may be invoked via the LDR 'RA+1' request or the following "control card" requests:

Load and Execute Library Program: libname,params

Load and Go: LGO,fname

The LGO request is identical to that described for CALIDOSCOPE in the CAL Guide.

Load Control: LDCTL,TSS
LDCTL,SCOPE

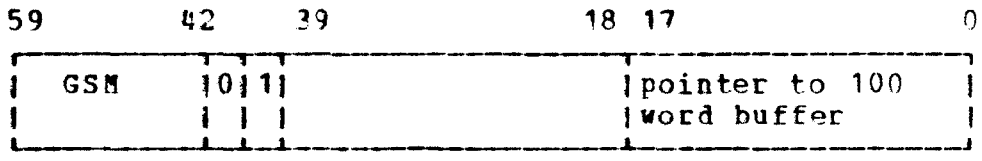
These requests toggle the loader between normal and the special TSS mode, in which all common blocks for each file are loaded are allocated at core addresses following all the program blocks. This sorting is useful for separating read-only and read/write sections of a subprocess. Note that the other LDCTL requests described in the CAL Guide are also acceptable.

Overlay Request: OVERLAY,fname

If the loader is in TSS mode, a core image is written onto the file fname without banner words (see Appendix A), i.e., just the contents of loaded and linked core. This mode is used primarily for producing subprocess descriptor files. If the loader is in 'SCOPE mode', a '0,0' level overlay is produced starting at cell 0 through the last word loaded.

Run Time Input

A running program can make a request for teletype input via the GSM 'RA+1' request:



The Simulator prints the up arrow and waits for input. The line typed in after the arrow is placed one character per word, right justified display code, zero-filled, in the buffer provided. The end of the line as typed is marked by a zero word.

Leaving the Simulator: FIN

The user exits from the Simulator by typing FIN after the >. When he does so, the program's log is appended to the file OUTPUT and all files are rewound and closed. Then the core file is deleted and control is returned to the Command Processor.

2.4.4 Error Messages

1. Request Errors:

COMMAND WORD > 10 CHARACTERS, TRY AGAIN.

A word in the request line exceeds 10 characters. The request is ignored.

I HAVE NO RECOLLECTION OF THAT FILE!

A file name in a PUT request has never been used before. The request is ignored.

I RECALL YOU HAVE ALREADY USED THAT FILE!

The filename in a GET request has already been used. The request is ignored.

NONSENSE COMMAND IGNORED, TRY AGAIN.

The request is not known and does not appear in the library.

2. CIO Errors:

BAD BUFFER PARAMETERS.

READ AFTER WRITE ATTEMPTED.

WRITE DURING READ ATTEMPTED.

Writing after a read is not implemented, except after an EOR.

UNKNOWN FCN CODE

The function code in the FET is unknown.

3. General Errors:

POINTER IS NEGATIVE.

POINTER > FL.

REQUESTED FL IS TOO BIG.

REQUESTED FL IS NEGATIVE.

UNKNOWN PP PROGRAM NAME.

4. Loader Errors:

FIELD LENGTH TOO SMALL TO LOAD PROGRAM

An address in a loader table exceeds the current field length. Loading is aborted.

FILE ENDS DURING TABLE

Before the word count of a loader table was exhausted, the end of the file was encountered.

FILE IS NULL

Loading of null files is not implemented.

IGNORED, NOTHING SUCCESSFULLY LOADED

An overlay or execute request could not be done because there is nothing loaded.

LDR FILE IS MIXED ABS & RELOCATABLE.

A given load must be either overlay or relocatable, but not both.

NO ENTRY FOR XFER LABEL FOUND.

The label on a COMPASS END statement cannot be matched to any entry point.

NO TRANSFER LABEL SPECIFIED. UNABLE TO START PROGRAM.

No starting entry point was specified.

TABLE ON LOAD ADDRESS < 0.

UNKNOWN TABLE TYPE.

YOU CAN*T LOAD A SYSTEXT FILE.

5. Other Errors:

BLOCK MISSING FROM A FILE.

This error can be caused by a null file or a file that was improperly written.

ERROR IN OCTAL NUMBER.

FNT FULL.

ILLEGAL USER XJ CALL.

The user attempted to call the SCOPE Simulator.

LINE LIMIT EXCEED.

USER CPU ARITH-ERROR.

6. Internal Errors:

SCOPE INTERNAL ERRORS - ZERO FNAME ON 1 FNT CALL.

SCOPE INTERNAL ERROR - NO FILE CAP ON FIO CALL.

Appendix A. File Structure

The CALIDOSCOPE fileset structure is simulated on TSS files by the Simulator. A TSS file is called a 'SCOPE file' when it has the following form:

> Individual logical records have a banner word at each end. These banner words contain the information necessary to scan through the file and access records.

59 56	53 49	35	17 0	file address
0 | 0	0	0	n	0
logical record n words level L1 (not EOF)				
4 | L1	20B	n	m	n+1
logical record m words level L2 (not EOF)				
4 | L2	20B	m	0	n+m+2
6 | 17B	30B	0	0	n+m+3
7 | 17B	1030B	0	0	n+m+4

The bits in the banner words are entered as:

59	EOR bit. Zero indicates a dummy banner.
58	EOF bit.
57	EOI bit. Set for the last banner in the file.
53-50	Level number of previous record.
49-36	FET code/status for previous record.
35-18	Record length of previous record
17-0	Record length of next record.

56-54 checksum, number of '1' bits in rest of banner mod 8

2.6 Running a BASIC Program

BASIC is called by typing BASIC to the ccmmand processor. When BASIC is ready to accept statements, it will type BASIC HERE, followed on the next line by its prompt character, a colon (:). To create a program, insert mode is entered by typing I followed by a carriage return. Now the lines of the program can be typed into the file. Note that, as in the Editor, it is necessary to be in insert mode to add lines to the program. If a line to be added has an error in it, BASIC will type an error message and that line will not go into the file. Instead, the line remains as the old line in the Line Collector so that it can be corrected before being put into the program. If a bad line is in a file that is read in with the R command of the Editor, the line does not go into the program, and it is typed after the error message. A line already in the program can be changed or edited, but if the new "corrected" line has an error in it, it is typed after the error message and the change will not be implemented. To aid in debugging, some PAUSE statements may be added. PAUSE stops execution with a message giving the next line to be executed, thereby allowing the programmer to check values with direct PRINT statements, or to enter any other direct statements. The program may be modified or changed in any way during a PAUSE. Unless the last line is an END statement, the program cannot be executed.

Execution of a BASIC program is started by typing RUN. All variables, functions, and arrays are made to be undefined and execution of the program starts at the beginning. If there are some variables which should not be destroyed because they have been set with direct statements or from execution of a previous program, execution can be started with a direct GO TO. When the program encounters the END or a STOP statement, it stops with the message EXECUTION COMPLETE.

The program may stop in the middle of execution for several reasons. If there is an error in running, such as division by zero or a jump to a non-existent line, an error message is typed out and execution is halted. Direct statements and/or editing requests can then be entered to discover and fix the problem. If, for example, the program jumped to a non-existent line, it could be fixed by adding a line which was forgotten or correcting the line number in the GO TO statement. After the problem has been fixed, the program could be restarted with a RUN statement or CONTINUE, which would restart execution with the line where the error occurred. Executicn can be halted by executing more lines than specified by a direct LIMIT statement. By using LIMIT, a program can be prevented from getting caught in a loop. If a program gets in a loop, it is possible to get out by hitting the panic button - CTRL-SHIFT-P. A message is printed as if an error had occurred. Execution can be resumed by typing CONTINUE.

There are two ways to leave BASIC. These are the same statements as are used to leave the Editor. Q means to quit and to destroy the

program that was created. F,fname saves the text of the program on the file specified so that it can be printed or loaded again later.

See PART FOUR - Languages and Processors: BASIC, for a complete description of the language.

2.7 Using BCPL under TSS

The compiler is invoked by typing BCPL to the Command Processor. The compiler then waits for lines of the form:

I=input;B=binary;C=compass;O=ocode;N=name;T;CR

All parameters are optional and may appear in any order. If the parameters input or compass are TTY, the input is taken from the teletype or the COMPASS program printed on the teletype, respectively The meanings of the parameters are as follows:

Parameter	Default Value	Use
I	TTY	Designates the file containing the source code to be compiled.
B	Binput	Designates the file on which the relocatable binary will be written. B=0 suppresses the output of binary. Default is BINPUT if I=INPUT or BBCPL if I=TTY.
C	0	Designates the file on which a COMPASS version of the program is written. This version may be assembled by COMPASS. C=0 suppresses COMPASS output.
O	OCODE	Designates the scratch file to be used for transmitting an intermediate object code between passes of the compiler.
N	same as I (BCPL if I=TTY)	Gives a name to the binary and/or COMPASS program produced; i.e., N=name would cause IDENT name to be the first line of the COMPASS program.
T		Causes compilation times to be printed.
CR		Check reentrant.

After each compilation BCPL waits for another line and exits when FIN is typed.

2.8 THE PRINTER DRIVER

Title:	Printer Driver
Code:	PD1.0
Author:	Keith Standiford, Computer Center, University of California, Berkeley
Date:	December 1970
Environment:	CDC 6400: Time Sharing System

2.8.1 Purpose

The printer driver is an interim program which allows the line printer to be used for listing long files as well as files containing lines with more than 72 characters. (If lines with more than 72 characters are printed on the teletype, characters past 72 will all be printed directly on top of one another.)

2.8.2 Usage

The printer driver is called by entering

PRINTER fname

where fname is the name of the file to be printed.

The printer driver should respond:

```
TSS PRINTER DRIVER VER 3.0
ENTER TITLE LINE
```

If instead the second line reads

FILE NOT SYSTEM STANDARD TEXT

it means that fname is either a non-existant (empty) file or that it was constructed under the SCOPE subsystem without asking for system standard text format.

The ENTER TITLE LINE should be answered with a line indicating the job number and name under which the output is to be filed in the output area.

The printer will respond to the title line by asking

SCOPE FORMAT CONVENTIONS?

to find out whether the first character of each line should be interpreted as a SCOPE carriage control character (see figure 1). A response line with a Y in column one is interpreted as a "yes". If Y

does not appear in column one, the printed file will be single spaced with an automatic page eject after 56 lines, and the first character of each line will be printed.

Figure 1. SCOPE Carriage Control Characters

Character	Vertical Spacing before Printing
blank	one line
0	two lines (double space)
1	top of next page (line 6)
+	no advance before printing[1]
-	three lines (triple space)
2	skip to next of lines 9,36 (next half page)
3	skip to next of lines 8,16,44 (next one-third page)
4	skip to next of lines 7,21,35,49 (next quarter page)
6	skip tc line 1 following concave paper fold
7	skip tc line 1 following convex paper fold
8	skip to line 1
X	single space and suppress automatic page eject until Y control character
Y	single space and resume automatic page eject
Z	single space and suppress automatic page eject for this line only

There is a lockout mechanism[2] fcr the printer since only one user can have the printer at a time. If the message EVCH CLEARED appears next, the user has the printer and his teletype keyboard becomes inoperative while the file is being printed. The message PRINTING ... should immediately follow EVCH CLEARED. If it does not appear in a few seconds, the printer driver has gotten into trouble. One should then take the termination exit below.

Occasionally the printer driver may type:

```
PRINTER NOT READY or HARDWARE ERROR or
PAPER OUT ON PRINTER
```

or words to that effect, to indicate that the printer itself is having trouble, e.g., is jammed, or out of paper. The printer driver will then type WAITING. Since nothing can be printed until the printer is made ready again mechanically, the user should notify an operator in the machine room (2-3043).

---

[1] Not implemented (1/71).
[2] The name of the lockout is PRNLOCK, OPERATE.

After about 30 seconds to a minute, the printer driver will check to see if the error has gone away. It will type PRINTING ... and attempt to continue. If the error still has not been rectified, the message will re-appear. For any message other than PAPER OUT ON PRINTER or PRINTER NOT READY, the condition, hopefully, is temporary and should go away. If this reoccurs several times, the user should notify the TA and take the termination exit described in the next paragraph. He should then attempt to print the file again (the printer lock may be lost; consult the TA). If the trouble persists, the operators should be notified of a persistant hardware failure.

If, however, the message

```
PRINTER LOCK ERROR, SORRY
ABORT
```

appears, the printer has detected that two printer drivers were attempting to run at once. This is possible only if someone has recovered the printer lockout mechanism. In this case, the printer driver aborts and returns immediately to the command processor. The user should attempt to print his file again.

If the printer seems to be taking much longer than it should, the process may be terminated with a panic (CTRL-SHIFT-P) followed by RECALL, which should return to the Command Processor.

Under normal conditions, when the file has been printed, the printer driver will return control to the Command Processor, which signals its presence by typing

```
COMMAND PROCESSOR HERE
!
```

and awaiting the next input line.

2.9 THE DISPLAY DRIVER

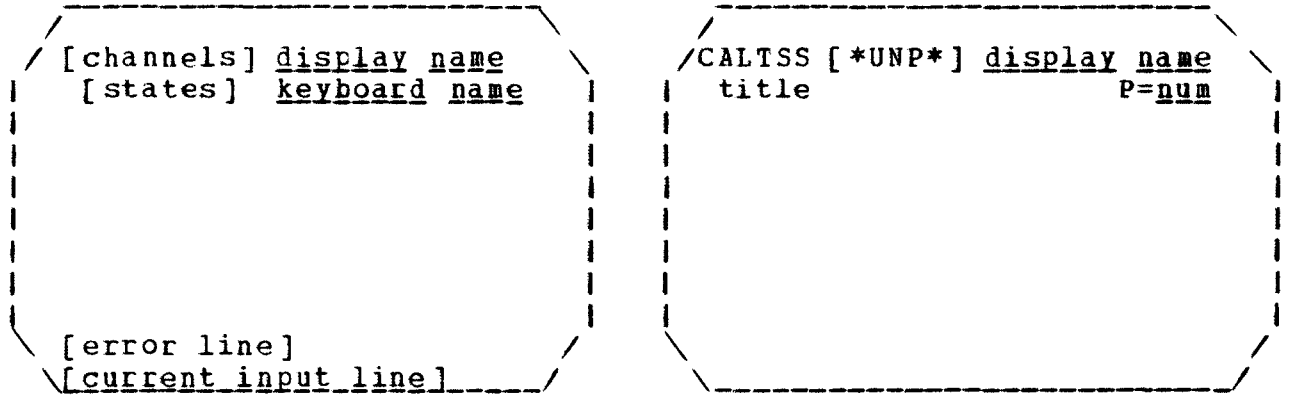
Title:	Display Driver
Code:	DD1.0
Author:	Keith Standiford, Computer Center, University of California, Berkeley
Date:	February 1971
Environment:	Machine: CDC 6000 Series Operating System: CALTSS Coding Language: COMPASS

2.9.1 Purpose

The display driver is designed tc allow the operator to display important information about the system as well as to use the displays for debugging.

The console consists of two CRT display screens and a typewriter-like keyboard. The CRT screens are divided into coordinates.[1] Approximately 40-50 coordinates at the top of each screen contain header information, while an equal number at the bottcm are reserved for displaying type-ins from the keyboard. The remaining center portion of the screens is left for the various displays. (See Figure 1.)

Figure 1. Display Screens



The right screen header contains the Time-Sharing System identification title in large letters, the "system unprotected flag", the right screen name, and the P-ccunter. The left screen header contains the channel statuses, the system state flags, the left screen name, and the current

---

[1] See Peripheral Equipment Manual.

logical keyboard name. The system state flags are interpreted as follows:

W	idle
U	user program running
S	system running for user program
P	ECS swap in progress
B	swapper calculations
I	interrupt code running

The lower portion of the left screen contains a line for error messages and one on which keyboard type-ins appear.

Three types of displays are available: CM/ECS displays for presenting the contents of areas in central memory or Extended Core Storage; user displays driven by programs inside the system and used for system-operator interaction; and the special M display, which provides ECS System statistics prepared directly by the display driver. Just as the console screens can present several types of displays, the console keyboard can be used to transmit messages to other programs besides the display driver. The term logical keyboard is used to describe the multiple functions of the actual console keyboard. It is anticipated that most of the system-operator interaction will occur via the logical keyboard arrangement.

In addition, the display driver keeps the real time and date, which are available to user processes via an ECS system call. In conjunction with the real time clock, the display driver provides a set of clocked event channels for processes desiring to wait until a specified time. (See 3.6 Event Channels.)

2.9.2 Method

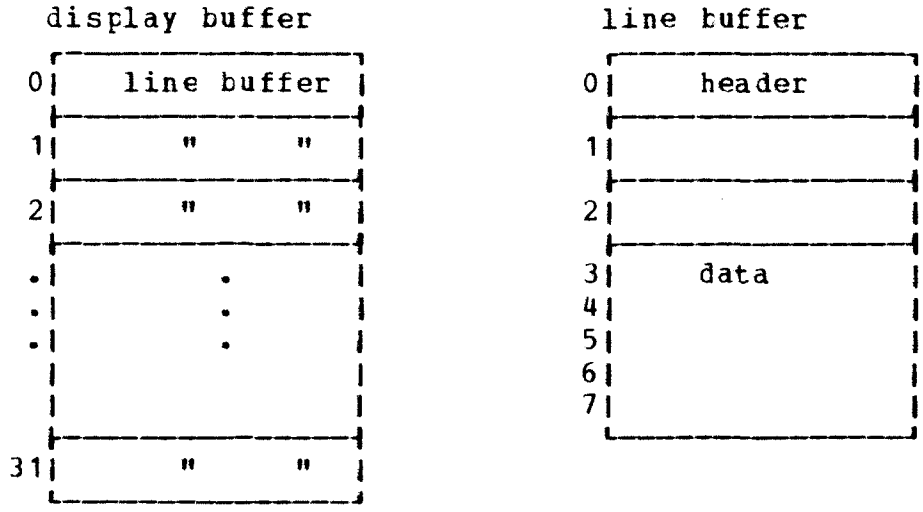
Assignment of display names is as follows:

A-D	CM/ECS displays
E-L	User displays
M	ECS system statistics

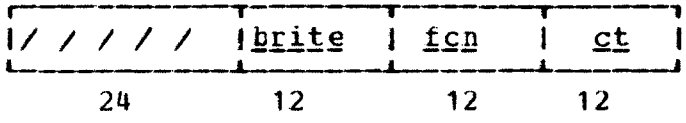
User displays are implemented by means of a display buffer (an ECS file) and an event channel. The display buffer consists of 32 eight-word line buffers which may be written into by a user program and will be copied onto the screen by the display driver when selected by the operator. Each line buffer has one header word followed by seven data words (see Figure 2). The intensity at which a message appears on the CRT screen is controlled by a parameter in the line buffer header word. Data from the line buffer are output verbatim, beginning with the high order byte of the word following the header word and continuing until ct bytes or the end of the buffer is reached. Data is assumed to contain positioning information and display code (or dot

coordinates if dot mode is selected). Note that a program using the display may write anywhere on the screen. Therefore care must be taken not to overwrite screen headers or the keyboard display area.

Figure 2. User Display Implementation



line buffer header word



brite is the relative brightness on the screen of the data in the line buffer. The normal range is from 1 to 5 with the brightness proportional to the value of brite. A value of 1 is the normal brightness, and a value of 0 is equivalent to a value of 1. If brite is 6, the message will appear at a brightness of 5 and will flash on and off about once a second. If brite exceeds 6, it will be assumed to be 1. Values for brite greater than 1 or 2 must be used sparingly; too many lines too bright will cause the screen to flash badly and the display driver to run slowly.

fcn is half of the function code sent to the device.[2] fcn=xyB where x is the mode and y is the character size:
x=1 indicates character mode
x=0 indicates dot mode

---

[2] See Peripheral Equipment Manual.

If x=1 then y=0 specifies 64 characters/line (small)
y=1 specifies 32 characters/line (medium)
y=2 specifies 16 characters/line (large)

Otherwise y is ignored.

Since x and y are each represented by three bits, only the lower 4 bits of fcn are significant.

ct is the length (in 12-bit bytes) of the data to be output. For example, if ct=35, 35 bytes are output; if ct=0, nothing is output.

After the file containing a user display has been changed, an event must be sent to the display driver to insure that the changes are reflected in its copy of the display, in case it happens to be showing that display on a screen. This event is a bit mask, with each '1' bit corresponding to an updated line. The lines are indexed 0-31 in the buffer, and the bits are numbered accordingly, with bit 0 at the low order end of the word. No response signaling completion will be sent to the user program.

The M display consists of the major system clocks, the current date and time (or the date and time from deadstart if never entered by the operator), some ECS system statistics, and some error statistics. The ECS system statistics currently consist of the ECS free space, ECS slop space, the number of allocated blocks and the number of free blocks. These statistics are subject to change as required by further development of the ECS system. The error statistics are displayed in the event of a system disaster.

A logical keyboard is implemented as a 12 word event channel which can hold the longest message that can be typed in at the console keyboard (64 characters) plus part of another message. Messages are sent 10 characters per event in left-justified display code.[3] As many events will be sent as are required to contain the message. The end of a message is signaled by a carriage return (60B) or by a "you lose" event if the buffer overflowed and part of the message was lost. The operator is also notified of lost data by an appropriate error message.

---

[3] The keyboard space bar maps into 00B in a message, rather than 55B (blank).

Note that there is no guarantee that the last event will contain zero fill following the carriage return. Note also that with this implementation, it is risky for two user processes to share a keyboard, since if two processes are hung on the same keyboard event channel, and a message greater than 10 characters is sent, each will get only part of the message. Capabilities in user displays and keyboards are located in the master device C-list. (See the section on Allocation.)

2.9.3 Usage

Keyboard Requests: Entries made by the operator are displayed in the lower left of the left screen. All lines beginning with the character '/' are interpreted by the display driver as display requests. Those not beginning with '/' will be sent to the current default keyboard. There is also provision for sending a message to a keyboard other than the default keyboard as well as for sending a message beginning with '/'. Blanks are ignored except in user messages and text commands such as DATE, PASS, and LOCK. If a mistake is made while entering a request, the backspace key erases characters one at a time, while the unlabeled key to the left of '=' clears the entire line. It is assumed throughout the following descriptions that all lines end in carriage return.

1. Send message to logical keyboard: /n.=msg

This request can be used to send a message to a logical keyboard which is not the current default keyboard. n is a legal logical keyboard name (the letters A-H) and msg is the literal text to be sent. If no user program desires a message from that keyboard at the time it is sent, the message is saved in a buffer (actually an event channel). When the buffer becomes full, a message indicating lost data will be given.

2. Change default keyboard name: /USE,n

This request changes the current default logical keyboard to the one specified by n, where n is a legal logical keyboard name.

3. Change display on screen: /l,r

This request changes the display shown on the left screen to l and on the right to r, where both l and r are legal display names (letters A-M). Either l or r may be omitted and the corresponding screen display will remain unchanged. If r is omitted, the comma may also be omitted.

4. Modify CM/ECS displays: /i,addr-expr

Core displays consist of four groups of eight consecutive Central Memory or ECS words each. The area displayed may be changed using the

"modify" display request. i and addr-expr are both address expressions, which are strings of octal digits optionally suffixed by one of the following characters:

C	for a CM address (default)
E	for an ECS address
+	for a positive number (ignored)
-	for a negative number (number is complemented)

An address expression is treated as a 60-bit quantity.

In the "modify display" request only C and E have meaning. i may be a number from 0 to 5 with the following interpretations:

0-3	set the ith word group to display the eight addresses starting with that given by addr-expr
4	set all four groups to display consecutive words starting at the address given by addr-expr
5	add the expression to the current starting address of each group; select repeat mode. Repeat mode causes the value of the expression to be added repeatedly to the starting address of each group, thereby providing a scan of memory. To leave repeat mode, the operator clears the message line. This may be accomplished by either pressing the clear key (the unlabeled key to the left of '=') or backspacing past the beginning of the message.

Note that where an address is not required, C and E suffixes are ignored. For example /4E,1 is treated the same as /4,1 while /4-,1 is illegal.

5. Operator date and time: /DATE,m/d/y
/TIME,hh.mm.ss

Date and time are entered in the above formats. The delimiters "/" and '.' are interchangeable. All numbers are expanded to two digits, if not specified as such, by adding leading zeros. No zero fields are allowed in DATE, and no trailing period is allowed in TIME, but otherwise no checks are made.

6. Clear and restore system protection: /PASS
/RSP

The system is normally protected. The operator may enter a three letter password after typing /PASS. If the password is accepted, the system unprotected flag will be set and *UNP* will appear on the right

screen header. It will remain until /RSP is entered to restore system protection.

7. Restore to initial state: /RESTORE

This request causes the display driver to return to an initial state defined at deadstart time. Although the operator should never need this request, is has proved useful in debugging since, in the event of a system crash, the display driver may be waiting for an event which will never occur, thus causing some part of its program to halt.

8. Lock and unlock CPU *: /LCCK
/UNLOCK

When /LOCK is entered, the display driver interrupt code will not give up the central processor until /UNLOCK is entered.

9. CM/ECS store *: /addr-expr = value

The contents of Central Memory or ECS may be altered with this request. Both addr-expr and value are address expressions as described above under 4. E and C are ignored for value, which is interpreted as a 60-bit word quantity.

10. Manipulate channel *: /reqnn[,mmmm]

There are several requests available for manipulating I/O channels. These are typed in the above general format, where req identifies the request, nn is a channel number (octal), mmmm is an output value (octal). n and mmmm will be right-justified, zero-filled by the display driver. The following requests are provided:

/ACNnn	activate channel nn
/DCNnn	deactivate channel n
/FANnn,mmmm	function mmmm on channel nn
/OANnn,mmmm	output mmmm on channel n

The channel status is checked, and these requests are refused with the error message NOT SAFE if conditions are unfavorable. In order to use these requests, display M must be showing on at least one screen. If it is not, the error message NEED SCREEN M appears.

System Disasters

A section of program within the display driver is devoted to checking for system disasters. When one occurs, it reinitializes the other programs in the display driver, places the M display on the right

---

* This request requires that the system be unprotected.

screen, clears the left screen and commits suicide. The M display then flashes disaster to indicate that the system is dead. It shows the address of the jump to DISASTER + 1, and, if the disaster was due to an ECS error, the absolute ECS address of the first bad word contained in the unsuccessful transfer is displayed.

The demise of the disaster check routine is followed within 100 msec. by the death of the clocked event channel driver when it tries to make clocked event channels tick. However, the console keyboard still accepts requests normally. Therefore, for instance, CM can be examined and modified. In fact, three possible courses of action exist:

1a. Nothing is done or the deadstart button is pushed - with no repercussions.

1b. The operator uses the display carefully and possibly discovers the problem.

2. The operator types a request for which the display driver requires help from the system. The display driver issues the message NOT SAFE.

   This will occur upon entering any of the following requests:

   Send message to logical keyboard
   DATE and TIME
   LOCK and UNLOCK
   ECS store

   Note: RESTORE may cause strange results since it reincarnates the disaster check routine.

3. The operator tries to display ECS, which may hang the display updater. This condition is not serious since the keyboard still responds. To correct the situation, the screen should be set to contain only Central Memory, then /RESTORE should be entered. For example,

```
/A                  (contains ECS; updater hangs)
/4,0                (contains CM only)
/RESTORE            (disaster check occurs again)
/A                  (desired results!)
```

3.1 FILES

3.1.1 File Structure

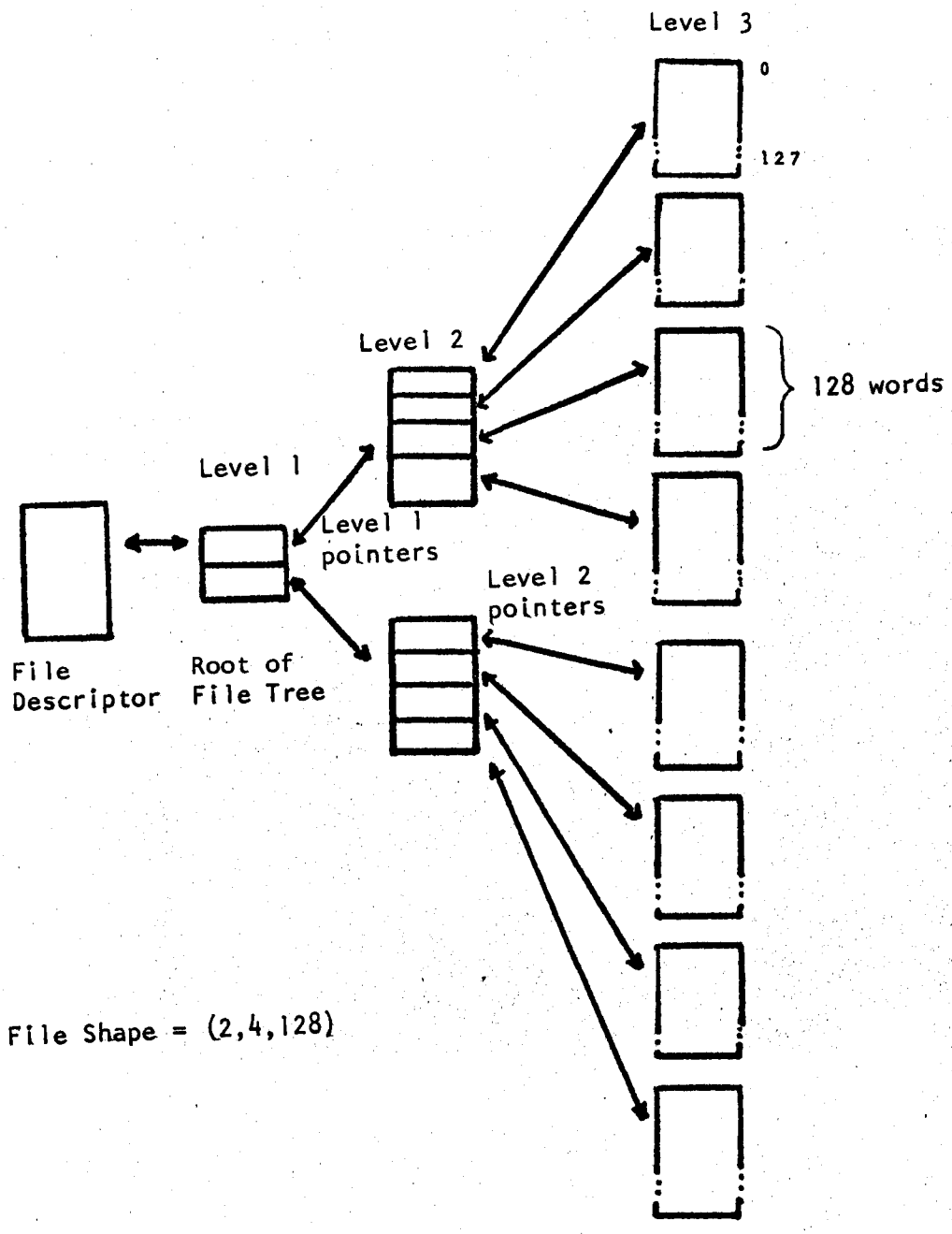
A file is a sequence of addressable 60-bit words used to contain information, such as program code or data. All CAL TSS files are constructed in a symmetrical tree structure (see Figure 1.) so as to permit a large file address space and, at the same time, to allow incremental allocation of file storage space. The addressable words of the file are contained in blocks of uniform length called data blocks which form the "leaves" of the file tree. The non-terminal nodes of the tree are called pointer blocks and contain links to either data blocks or other pointer blocks.

For any file, there is a sequence of positive integers which are specified when the file is created and which describe the shape of the file tree. The first number in the sequence is the number of branches extending from the root of the tree. Each successive integer in the sequence is the number of branches from each non-terminal node (pointer block) in the file tree at the corresponding level. The last shape number is the uniform size of the data blocks. All shape numbers, except the first one, are required to be powers of two to facilitate file address decoding. Since the tree is symmetrical, these numbers completely describe its shape, and their product gives the total number, n, of addressable words in the file. These words are addressed from top to bottom using consecutive integers ranging from 0 to n-1. For instance, if the shape numbers are 2,4,128, there are two branches extending from the root of the tree, each pointer block at level two branches to four data blocks, and the tree contains eight data blocks of 128 words. The four data blocks attached to the left side of the tree contain addresses 0-127, 128-255, 256-383, 384-511, respectively, and those on the right side contain addresses 512-639, 640-767, 768-895, and 896-1023.

Use of the tree structure means that, even for very large files, the need to allocate contiguous file storage space is limited to the file's data block size. In addition, while the maximum size of a file is specified when it is created, unused or non-existant portions of the file are not allocated space until they are needed.

Whenever necessary, therefore, data blocks can be added to (or deleted from) a file at any file address specified by the user. Appropriate accounting is performed to charge (or stop charging) for the space occupied by the block and to control (i.e., limit) use of the storage space. Pointer blocks connecting the data block to the root of the tree are supplied when necessary. Newly created data blocks are initialized to zeros. Data blocks may be moved from one file to another, if the data block sizes are the same, and thus data blocks already containing data may be added to a file.

Figure 1. File Tree



3.1.2 Disk files

Since CAL TSS uses ECS (Extended Core Storage) as its primary memory and since ECS space is limited, files in CAL TSS may reside either on the disk or in ECS. In fact, the disk is used for permanent file storage, and disk files which are active, or open (see below), have a file incarnation in ECS which has the same structure and shape as the disk version. Portions of the file (i.e., data blocks, corresponding to file address ranges) are copied between the disk and ECS and are added to or deleted from the ECS incarnation of the file according to the requirements of the user process. This procedure is referred to as "attaching" or "detaching" a block.

For a user process to read (write) a file requires use of the "file read (write)" action, which transfers words between the address space of the running subprocess and the data block(s) of a specified file in ECS. The user indicates the address in the file of (for) the desired information, the address in Central Memory of the area to be read into (written from), and the number of words to be read (written). If the data blocks containing the required range of addresses have already been attached to the ECS incarnation of the file, the transfer of information is performed in a straightforward manner by the ECS level of the system. If, however, one or more of the data blocks are not present in ECS, the transfer proceeds until the first non-present file address is encountered, whereupon F-return action (see "process control") passes control to the disk subsystem. Here the entire transfer is performed by 1) copying or re-copying blocks which were already present in ECS and 2) attaching the blocks which were not present in ECS, copying these blocks, and detaching them again. This attach, copy, detach process is performed for each required data block which was not present in the ECS version of the file. Required data blocks which were present are transfered with normal ECS level file read (write) actions.

The contents of each subprocess address space are specified by the subprocess map (see "subprocesses"). The map establishes a relationship between sections of the subprocess address space (memory) and portions of ECS files. Whenever a subprocess is swapped into central memory, the appropriate portions of the files designated by the subprocess map are copied from ECS into central memory. When the subprocess is swapped out to ECS, the reverse operation is performed (except for map entries which are "read only"). Portions of disk files which are to be used in a subprocess map are attached to the process and held in ECS as long as the map entry remains in force.

3.1.3 Disk File Open-Close

In order to keep track of which disk files are in use and therefore must have an ECS incarnation and file control information (file header block) in ECS, programs must explicily open and close disk files. In this way, the system can maintain a ccunt of how many processes are using any particular file. Whenever a process intends to use an existing disk file, it must "open" the file by presenting a disk file capability for the file. A disk file capability contains in its 2nd word the disk file unique name and the disk address of the file header word. The disk system will return an ECS file capability for the ECS incarnation of the disk file. When a prcgram is through with a file, it should "close" the file. Although the open and close actions normally occur in pairs, it is possible, and in some situations advantageous, for a process to open a file more than once before ever closing it (e.g., separate subprocesses using the same file). If only one subprocess at a time used a file, the disk subsystem could simply maintain an open counter which it incremented each time an open was issued and decremented when a close was issued. A zero count would indicate that all processes were through with the file. However, since more than one subprocess can use a file at a time, the disk system maintains both a local open-close counter which keeps track of the opens and closes by a single process, and a global open-close counter which indicates how many different processes are using the file.

When a process opens a disk file, the lccal open count for the file is incremented. If the local open ccunt went from zero to one, the global open count is incremented. If the glcbal open count went from zero to one, the file header block is brought from the disk into ECS, and an ECS incarnation of the file is created. This procedure is reversed for a "clcse" action. Whenever the global count goes to zero, the disk system deletes the ECS incarnation of the disk file after updating the contents of the disk versicn of the file.

While a disk file is open and being manipulated, care is taken to insure that the versicn of the file on the disk is readable at all times. Data blocks of a file which have been modified are written at new "swapped" locations on the disk leaving the original "fixed" version of the file unmodified. When the file is finally closed by the last process holding it open (or upon request for a "pseudo-close"), pointer blocks are written to new locaticns on the disk to reflect the new locations of the modified data blocks. After all pointer blocks, except the root block, have been successfully written, the root block is re-written at the same disk locaticn to tie the modified portions of the file to the root (or header) block. This mechanism permits the system to write file data blocks at the first available disk position and preserves the cld contents of the file in the event of a system failure. The system may, at any time, initiate a "pseudo-close" to reclaim space on the disk by causing the file contents to be updated.

Another mechanism is also available to the user who must insure that the contents of his file on the disk are updated only after a sequence of changes to the file have all been completed. The contents of two files can be interchanged without doing any copying. The interchange is implemented in such a way that one of the files will either remain unchanged or have the contents of the other file regardless of a system failure at any time.

3.1.4 Data Block Attach-Detach

When a portion of a file is used for reading or writing or is in a map entry, it must exist in the ECS incarnation of the file. When a data block is first "attached", it must be copied from the disk to ECS. This transfer is accomplished in two steps in order to avoid a time interval during which the data block exists in the ECS file but does not have the correct contents. A holding file in ECS is used so that during an attach, the information is copied from the disk to a block in the holding file. Once this copy is complete, the block is moved to the proper block of the ECS incarnation of the disk file. User programs can attach a block of a file whenever the block is needed or will be needed soon. Placing a portion of a disk file in a subprocess map also forces the block to be logically attached. An attach count mechanism similar to the open count mechanism is employed for each block to determine when a block can be removed from the ECS incarnation of the disk file.

When the user is through with a particular portion of a file, the block(s) can be "detached". The local attach count for each data block is reduced. If the local count becomes zero, the global count is reduced. If the global attach count is exhausted, the block is removed from ECS. Associated with each data block of an ECS file is a "dirty-bit" which is set whenever the block is written into or placed in a read-write map entry. If the dirty bit is not set, the block does not need to be written back to the disk since it has not been modified; it can simply be deleted. Otherwise, the block is moved to the holding file and then copied to the first available position on the disk.

3.1.5 Disk File Interlocks

The user may protect access to any particular file by other users using mechanisms in the directory system. He may also choose to share particular file(s) with other selected users on a read-only or read-write basis.

Although the open-close counters keep track of file usage, only the special "exclusive open" action places any restrictions on the number of users who may open the same file for reading or writing at one time. Thus it is possible for two or more processes to be writing into the same portions of a file simultaneously, obviously a precarious situation. To aid interprocess cooperation, the disk system provides a

completely voluntary claim mechanism. If a process wishes to coordinate its use of a file with other possible simultaneous users it can make a "claim" on the file. If the user plans only to read the file, he may make a shared claim. More than one shared claim may be honored at the same time. If, on the other hand, the user wants to write on the file, the process should make an exclusive claim for the time during which it will be altering the file. An exclusive claim will not be honored until there are no other claims in effect on the file and, while an exclusive claim is in effect, no other claims, exclusive or shared, may be honored for that file. A queue of claims waiting to be honored is maintained on a first-come, first-served basis when claims on a file come into conflict.

3.1.6 File Accounting

Charges for file usage are based on the amount of ECS and disk space occupied by the portions created and/or used. Parts of a file which remain in ECS as long as the file is open, such as the description of the file shape and the data block of a single level file, occupy fixed ECS space. Data blocks of multi-level files which have been attached by one or more processes occupy swapped ECS space. Even though several processes may be using the same file, each is charged for the ECS space occupied by the portions it has attached and for the fixed ECS space. The disk space occupied by a file is charged to the funding directory (see "accounting and allocation") which is associated with the file at the time of the file creation. Temporary space occupied by data blocks which have been written to the disk is not charged to the file. Also, for open disk files, the amount of Disk System Storage used by the file is regulated.

If a disk file is to be used by several processes, it may be "frozen". When a process "freezes" a file, all existing file blocks are attached and the ECS space occupied by the file is charged to the process. Processes which open the file after it has been frozen are not charged for any fixed or swapped ECS space in connection with the file. Thus commonly used files can be frozen so that only one process pays for the file space (which is charged as fixed ECS space).

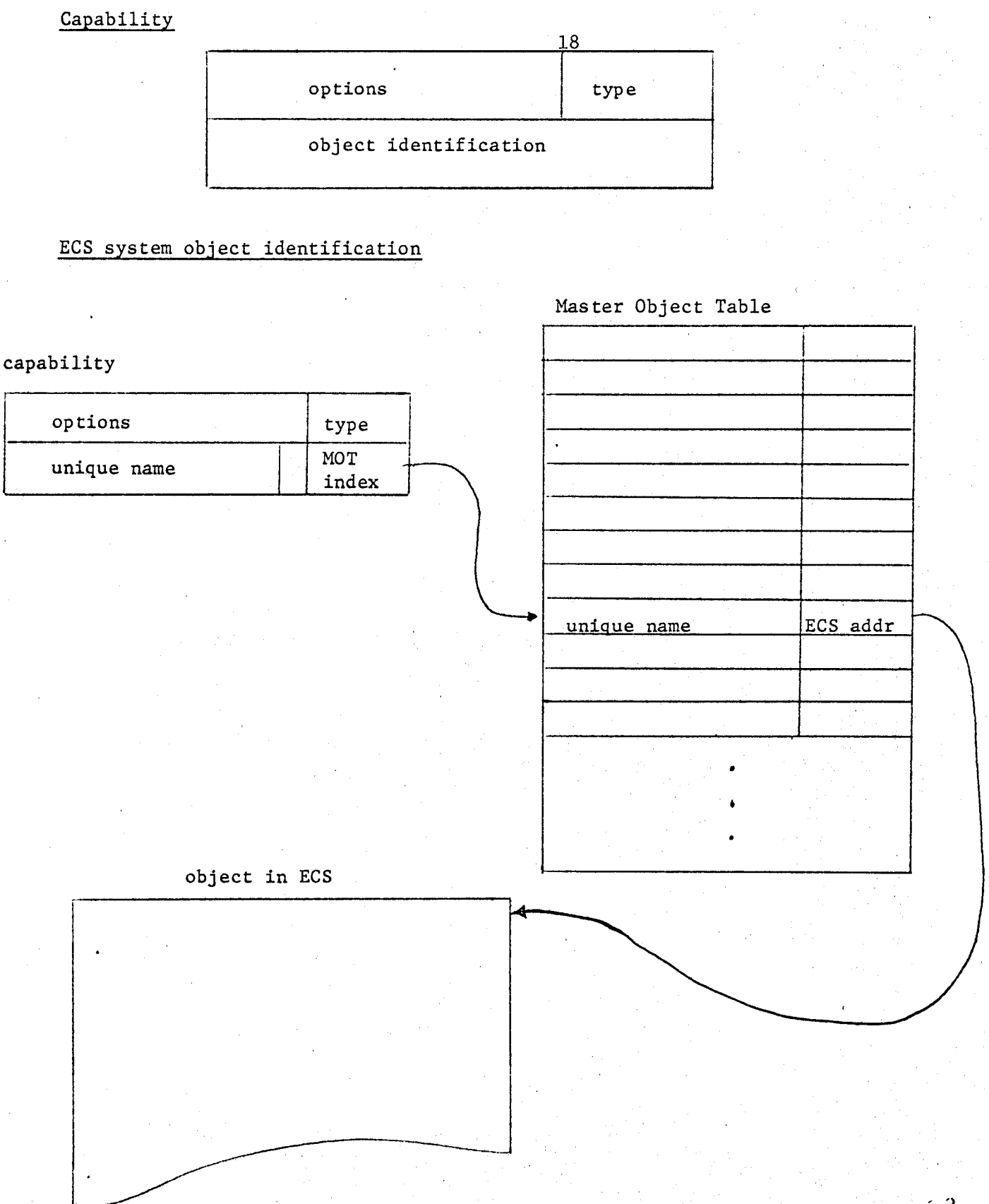
3.4 CAPABILITIES AND C-LISTS

3.4.1 Capabilities

Within CAL TSS objects are identified, and access to them authorized, by means of capabilities. A capability identifies an object by specifying the type of the object and providing a 60-bit datum which uniquely identifies the item. The interpretation of the datum depends on the type of the object. This 60 bit datum, used to identify the object named by the capability, is called the "protected name". The protected name along with the type are used to identify and access the object referenced by the capability. The information content of the protected name cannot be modified once it is set into a capability and thus can contain critical information, such as a pointer or a sequence number, to be used in accessing or identifying the object. In some cases, when the object being named by the capability is very small, the object itself may be stored in the protected name (e.g., class codes, capability creating authorizations, and access keys). Each capability also carries 42 "options", which are used to indicate the particular kinds of access permitted to the object represented by the rest of the capability. Each option represents a specific operation that can be performed on that object; for example, in a file capability, options are designated for reading, writing, deleting and other more obscure operations on files. When a capability is copied, options may be turned off, thereby producing weaker capabilities (ones allowing fewer operations) from stronger ones.

Each capability is two (60-bit) words long. See Figure 1. The first word contains the 18 bit type field and the 42 options. Objects which are recognized by the ECS system and occupy space in ECS (i.e., files, event channels, processes, C-lists, operations, and allocation blocks) are identified in the second word of a capability by a unique name and an index into the Master Object Table (MOT). All access to these ECS system objects is through the MOT, and includes checking the unique object identification (assigned when the object was created, and stored in the MOT entry). Also contained in the MOT entry for an object is the single pointer to the start of the object in ECS, which makes possible the compaction of storage in ECS without the necessity of updating capabilities referring to the objects. On every reference to an object, the unique name assigned to the object is checked against the unique name stored in the MOT. This scheme makes it possible to destroy an object and re-use its MOT slot without locating and invalidating all capabilities referencing the object.

Figure 1


Capability
18
options
type
object identification
ECS system object identification
Master Object Table
capability
options
type
unique name
MOT index
unique name
ECS addr
object in ECS

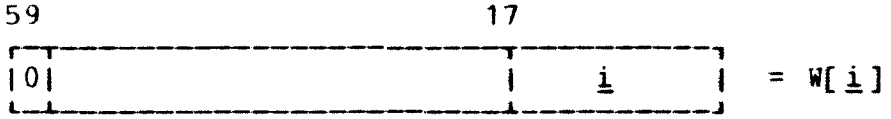
3.4.2 Capability lists

Since capabilities are used to authorize access to objects within the system, it is necessary that the user be prevented from fabricating them without proper authority. To this end, capabilities are gathered together into Capability-lists (C-lists) which are themselves objects within the system. A capability-list is a sequence of capability slots which are indexed from 0 to n-1 (where n is the length of the C-list and is fixed when the C-list is created). "Empty" capability slots are represented by a pair of zero words. Capability lists may be thought of as a peculiar sort of addressable memory which can be accessed only through the system. To modify the contents of a C-list, there are system functions which copy capabilities from one place to another, clear a capability location, or return a capability for a new object.

To provide an initial set of capabilities for programs running on the system, every process has a distinguished C-list called its working C-list. A capability is referenced by specifying an entry in the working C-list. If we let W denote the working C-list, then W[i] denotes the capability in the i-th slot of W. Since C-lists are objects which can themselves be named by capabilities, it is possible to specify capabilities in more complex ways; e.g., W[i][j] would be the j-th entry of the C-list named by the capability in the i-th entry of W.

In the interest of simplicity, all capabilities referenced as parameters of operations are allowed to invcke only one level of indirection. Thus, two formats are recognized for specifying capability parameters.

Direct capability reference:



Indirect capability reference:

By these mechanisms of capability referencing and modification, it is clear that a capability (in a C-list) is the protected name of the object it represents; while the direct or indirect reference (i.e., its address in the C-list) is the unprotected name of the object named by the capability. The integrity of the protection system is assured

because only capabilities reachable from the working C-list of a process have unprotected names in that process. The fact that a capability is in a process' working C-list is construed as evidence that the process has the right to access the object named by the capability.

3.4.3 Capability Creating Authorizations

Since the presence of a capability in a C-list is interpreted by the system as a-priori authorization to manipulate the object identified (named) by the second word of the capability, the user must never be allowed to directly fabricate a capability and place that capability in a C-list. Functions within the system which "create" objects are permitted to fabricate capabilities for them and place them into C-lists. Since the interpretation of the second word (identification or protected name) of a capability is dependent upon the type of the object, the injunction against fabricating capabilities can be relaxed somewhat. In particular, the privilege of fabricating capabilities for a particular type of object could be authorized if the type were known to be different from any other type already being used.

By extending the number of types of objects in this way, a subsystem may represent objects which it creates and maintains by capabilities with an extended type. To achieve this flexibility and extendability while retaining the protection and access-control features of capabilities and C-lists, a new object is implemented in the basic (ECS) system. This object, called a Capability Creating Authorization (CCA), is simply a protected name for the type which is being authorized. The identification of the new type is carried in the second word of the capability for the capability-creating authorization. The ECS system, when creating a new CCA, simply fabricates and returns a CCA capability naming the next 18-bit integer satisfying the contraints of a type number.

The ECS system, when presented with a CCA capability and a 60-bit datum, will fabricate and return a capability of the type named by the CCA with the second word of the new capability set as specified by the supplied 60-bit datum. Thus the new capability is the protected name of whatever the identificatiCn or naming portion of capabilities of this type represent. This mapping (i.e., the meaning of the protected name) presumably is applied by the subsystem which creates the extended type capabilities.

3.6 EVENT CHANNELS

3.6.1 Description

Event channels are ECS objects which are used to synchronize the behavior of running processes as well as to implement "block" and "wake-up" mechanisms. Events can be sent to or received from an event channel. An event consists of two 60-bit words: the first identifies the sending process or the sending channel while the second is a 60-bit datum, presumably carrying information for the process which receives it.

The event channel is composed of two queues, one for events waiting to be picked up by some process and the other for processes which are waiting for an event to arrive. At least one of the queues is always empty; either there are more processes requesting events than there are events (event queue is empty), or more events have been sent to the channel than there are processes requesting events (process queue is empty). Of course, the channel may be idle, and both queues will be empty. Since the event queue is stored in the event channel, it has a maximum size which is specified when the event channel is created. The length of the process queue, however, is unlimited since it is maintained by means of a pointer chain through the processes in the queue.

A user process can create an event channel, send an event to a particular event channel, request an event from one event channel or a set of them, and destroy an event channel. (See Event Channel Actions.)

3.6.2 Sending Events

When a user process sends an event, the event (event datum provided by sender) is passed to the first process in the process queue if there is a process waiting. In this case, the first waiting process is removed from the process queue, passed the event in X6 and X7, and scheduled to run. If the process queue is empty, the event is placed in the event queue if there is room. Should there be only one free slot in the event queue when an event is sent, the intended event datum is replaced by a special "you lose" datum so that the process which eventually gets the event will be aware that the event queue became full and that information may have been lost. If the event queue is full, the event cannot be sent. The system always returns a flag to the sending process indicating the disposition of the event (i.e., event passed to a waiting process, placed in event queue, "you lose" event placed in event queue, or event queue full).

3.7.3 Getting Events

A user process may attempt to get an event from an event channel. When an event is available, it is immediately delivered to the process in X6 and X7. If no event is available, the process may elect either to "block" or to be notified immediately that the event queue is empty by means of an F-return (see Section 3.3). If the process "blocks" (i.e., wishes to wait until an event arrives), it is added to the end of the process queue of the event channel and "descheduled". The process will not execute any more instructions until enough events have arrived on the event channel to bring the process to the head of the process queue. A process waiting on the process queue of an event channel may also be restarted by an "interrupt" to the process which has sufficient priority to pre-empt the program which hung on the event channel (see "interrupts"). The next event after the process becomes the head of the event queue will cause the process to be scheduled to run, and the event will be passed in X6 and X7.

A process may also try to get an event from one of a list of event channels. The event channels are checked one at a time and the first event from the first non-empty event channel on the list is delivered to the process in X6 and X7. If all of the channels are empty, the process may elect either to be notified immediately by an F-return as in the one channel case, or to wait for an event. In the latter case, the process is queued on all the event channels on the list. The first event to arrive on any one of the channels (which does not have earlier processes waiting) is sent to the process, which is simultaneously unqueued from all the event channels.

The number of event channels which may be interrogated simultaneously is limited by a parameter built into the system. It is further restricted by a parameter supplied when the process is created.

The channel which delivers an event is identified in the first word of the event. If the process receiving the event was hung on a list of event channels, the ordinal of the sending channel is packed into the scale. If the process was only hung on one channel, it gets a 1 packed in the scale.

3.6.4 Timed Event Channels

A set of special event channels, called clocked event channels, provide facilities whereby a program may economically wait for varying periods of time. Each of these channels is set for a specified frequency. When one of them ticks, all processes hung on the channel receive an event to activate them. (The event datum is the reading from the master (micro-second) clock at the time the tick occurred.) These channels are implemented so that they tick on even multiples of the time of day clock, which is entered by the operator when the system is

initialized and kept by the display driver. Channels are provided which tick whenever the time of day clock turns over to each even:

1/10 second
1 second
10 seconds
1 minute
10 minutes
1 hour

For instance when 10:00:00 turns over, all the channels tick but at 10:00:01 only the 1/10 second and one second channels tick; at 10:00:10 these two and the ten second event channel tick. There are two things to note: First, in order to time something using these channels, one must wait for an initial tick, and then hang again, or get the time and date from the system and compute the interval until the next tick should come. Second, these channels run off of the real time clock, which is entered by the operator. Consequently, whenever the time is entered, the next tick of each event channel will not be at the expected interval from the previous tick.

3.6.5 Uses for Event Channels

Event channels may be used for such things as inter-locking two or more processes, free list allocation, and sending messages. For example, if the system has only one printer, only one user may access it at one time. If access to the printer is governed by an event channel with only one event, users wanting to print will try to get the event from that channel. If some user already has the event, others must hang in the process queue until the user is finished printing and sends the event back to the event channel.

Free list allocation can be handled by an event channel if the single event is a pointer to the head of the free space list in which the head of each block of free space points in turn to the head of the next block of free space. When a process needs space, it gets the event from the event channel, takes the pointer from the head of the list and puts it into an event which it sends back to the channel; when it is finished with the space, it gets the event, places the pointer in the event into the first word of the space being released, puts the address of the first word of the space being released into the event, and sends the event back to the channel.

Messages can be sent between processes via an event channel. The message might consist of one or more events, 10 characters per event with a special character to signal the end of the message. These events are sent sequentially to the event channel, and, if the receiving process is not waiting to get them, are stored in the event queue. Note that it is vital to the proper function of this particular

mechanism that only one process at a time be getting or sending events from the channel.

File Actions

The Time-Sharing System provides a number of actions for handling files. A description of files and how they are handled under TSS appears in Section 2.1. The following list summarizes the available actions:

Create a disk file via directory system
Create an ECS file
Create a disk file
Create an ECS file data block
Create disk file data blocks
Delete a data block from an ECS file
Delete a block from a disk file
Delete an ECS file
Delete a disk file
Move an ECS file block
Open a disk file read-only (read/write)
Exclusively open a disk file
Close a disk file
Attach file block
Detach disk file blocks
Interchange file contents ("shazam")
Read (write) files
Make an exclusive claim on a file
Make a shared claim on a file
Release claim on a file
Read shape of an ECS file
Display disk file status
Display status of n-th disk file
Display n-th attached block of disk file
Check for missing ECS data blocks ("probe")
Check for missing disk file blocks
Close all open files
Set-reset close all over-ride
Test and reset dirty bit
Return disk subsystem clock

Detailed descriptions of these actions appear below. For a general description of how the user calls the system to initiate an action, see Section 3.5.

4.1.1 Create Disk File via Directory System

(To be supplied)

4.1.2 Create an ECS File (EC:CFIL)

IP1 C: Capability for allocation block (OB.CRFIL)
IP2 D: C-list index to return capability
IP3 D: Number of levels in the file
IP4 D: Pointer to a list of shape numbers

When a file is created, only the file descriptor is constructed. The file descriptor contains a pointer to the root of the file tree (initially empty since no data or pointer blocks exist). The user supplies the capability of the allocation block which is to fund the ECS space occupied by the file. The user must also supply the index of the C-list slot where the system will put the capability for the newly created file (all option bits in the capability for the new file are turned on). The last two parameters of the file create action, indicate the number of levels (n) contained in the structure of the file tree, and a pointer to a list of n shape numbers (S1 through Sn), the first n-1 of which indicate the number of branches from each pointer block at each successive level of the file tree; the last shape number (Sn) gives the uniform size of all data blocks in the file. A "one level file" (IP3=1) consists of a single data block of length S1. Each shape number (S1 excepted) must be a non-negative power of two.

Possible errors while creating an ECS file:

Class	Number	Description
E.ABLOCK	E.NOABLK	Allocation block does not exist
E.ABLOCK	E.NOECS	No ECS available
E.PARMS	E.NEGIX	C-list index is negative.
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPT	Pointer to list of shape numbers is negative
E.PARMS	E.NEGPAR	Level number n < 1
E.PARMS	E.BIGPAR	Level number is too large or Pointer to list of shape numbers plus list length exceeds user's FL
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NEGSIZ	Non-positive shape number
E.FILES	E.BIGSIZ	Shape numbers exceeds $2^{17}-1$
E.FILES	E.NOTPOW	Shape number other than S1 not a power of 2
E.FILES	E.BIGFIL	Total size of file exceeds $2^{59}-1$

4.1.3 Create a disk file [1] (DF:CFIL)

Input parameters:
IP1 D: Disk accounting record number
IP2 BD: Shape numbers S0 through Sn

Returned parameters:
RDAT : Disk unique name, header block size, and disk address (oneword)
RCAP : ECS file capability for new file

A file of the specified dimensions (see 'Create ECS file') is created in both ECS and the disk system data structure. The resulting disk file is opened for the creating process. The new file is associated with the disk accounting record specified by the first parameter (IP1), and this record must fund all permanent disk space occupied by the file. Although space is reserved on the disk for the file header block (see below), it is not copied to the disk until the file is closed.

For a disk file, the first level of pointers (or the data block in the case of a one level file) is associated with the header block of the file (in contrast to the scheme for ECS files in which the file descriptor and file root are separate). Thus, if the new file is a one level file, the data block is funded and added to the ECS incarnation of the file.

An ECS file capability (with write access) together with the disk unique name and header block address are returned to the caller via the return parameter mechanisms (see 3.5). The ECS file capability has all options set except for: OB.DSTRY, OB.CHNAM, OB.CREBL, OB.DELBL, OB.PLMAP, OB.FDAE, and OB.TRDB. There are many restrictions on the shape of a disk file. Considerations of efficiency and utility have forced what may seem to be arbitrary limitations on the range of the dimensions of a disk file.

Shape restrictions: one level files - data block size < 512 - 4

multi-level files - number of levels $\leq$ 11
data block size = 128 or 256 or 512
pointer block fan out $\geq$ 8
$\leq$ 128
= 2**n
(for some n)
first shape number (S1) $\leq$ 128
total length of file $\leq 2^{36} - 1$

---

[1] Privileged operations are used by privileged system subprocesses and are not allowed to be in user C-lists.

In addition to errors which may be detected while processing the file description, errors may be encountered in funding the newly created file. Fixed ECS space is funded from the process allocation block: a one-level file receives the data block size plus 4 words; a multi-level file receives the sum of the number of levels plus the first shape number plus four. Permanent disk space is funded from the disk accounting record (IP1); a one-level file receives from 1 to 7 sectors depending on the data block size, and a multi-level file receives one sector. Enough Disk System Data Storage (DDS) is reserved to permit creating at least one block on the file. Additional DDS space may be reserved by making the appropriate call upon the disk system.

Possible errors while creating a disk file:

Class	Number	Description
E.PARMS	E.NEGPAR	Number of levels < 1
E.FILES	E.LLEV	Number of levels too large
E.FILES	E.NEGSIZ	Data block size negative or too small (< 128) - multi-level file Shape number too small (< 8) - one level file
E.FILES	E.BIGSIZ	Data block too big (multi-level > 512) (one-level > 508) Shape number too big (> 128)
E.FILES	E.NOTPOW	Data block size or pointer block size (multi-level file) not power of 2
E.FILES	E.BIGFIL	File too big (> $2^{36} - 1$)
E.FILES	E.NOLFH	Too many locally open files
E.FILES	E.FULL	Disk system tables full
E.ABLOCK	E.NOABLK	Disk accounting record does not exist
E.ABLOCK	E.NOECS	Not enough fixed ECS for file
E.ABLOCK	E.NODSK	Not enough permanent disk space in disk accounting record
E.ABLOCK	E.NODDS	Not enough DDS space

4.1.4 Create an ECS File Data Block (EC:CBLK)

IP1 C: Capability for file (OB.CRFBL)
IP2 D: Address of block in the file

Once an ECS file has been created, data blocks of the declared length (Sn) may be added subsequently, one at a time, to hold data or code. A count of the subprocess map entries which reference the data block is maintained with each data block. (This count is important when deleting a block - see below). To create a block, the user supplies a capability for the file to which the block is being added, and an address which is contained in the block which is to be added to the file.

When a data block is added to a file, it may also be necessary for the system to create some or all of the pointer blocks between that data block and the file descriptor. Recall that pointer blocks are required to link the file descriptor to the data blocks in any file with more than one shape number (i.e., not a one level file). All newly allocated ECS space is charged to the allocation block associated with the file.

Possible errors while creating a block:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPT	Address of new block is negative
E.PARMS	E.BIGPT	Address of new block $\geq$ file length (address range: 0 to length-1)
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NOFIL	File does not exist
E.FILES	E.ISBLK	Address of new block corresponds to an already existing block
E.ABLOCK	E.NOECS.	No ECS space available

4.1.5 Create disk file data blocks (DF:CLBK)

IP1 C: ECS file capability for locally open disk file (OB.DCRBL)
IP2 D: Starting file address
IP3 D: Word count

Data blocks may be added to any locally open disk file. The file to be enlarged is specified by giving the ECS file capability (IP1) for a locally open disk file (i.e., a disk file which has been opened by the calling process). More than one block may be added at once. Blocks beginning with the one containing the starting file address (IP2) through the one containing the starting address plus the word count (IP2 + IP3) are created and attached to the calling process.

All new data blocks and any necessary new pointer blocks are funded on the disk by the disk accounting record associated with the disk file. Disk space for all data blocks is funded before any blocks are created (unused space is returned in case of an error). Swapped ECS space for attaching the new data blocks is charged to the swapped ECS account of the calling process. The actual ECS file space occupied by the new file blocks is absorbed by the disk system ECS allocation block.

The create disk file data block operation will generate an F-return if the disk file to be enlarged is frozen, (i.e., all usage is being paid for by a single user). On one-level disk files this action returns an error after checking that the file addresses are in range. This is because the data block of a one-level file always exists and can neither be created nor deleted. Errors may occur after one or more blocks have been created. The file address of the first block not created is added to the error number to indicate the state of the file when the error occurred.

Possible errors while creating a disk file data block:

Class	Number	Description
F.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPAR	Negative starting file address or negative word count
E.PARMS	E.BIGPAR	File address plus word count exceeds file length
F.OPER	E.CAPTY	Type or options bad
E.FILES	E.NODFIL	No such disk file opened by this process
E.FILES	E.DIOERR[2]	Error on pointer block read from disk (modifier = file address of first block not created)

---

[2] Error may occur after zero, one or more blocks have been created.

E.FILES	E.ISBLK[2]	Data block address already exists (modifier = file address of block which exists)
F.FILES	E.NABR[2]	No attached block record, i.e., disk system out of local storage (modifier = file address of first block not created)
E.FILES	E.ZLEV	Attempt to add a block to a one-level file
E.ABLOCK	E.NODSK	Insufficient disk space to fund all data blocks
E.ABLOCK	E.NOSWP	Insufficient swapped ECS in process disk accounting record.

4.1.6 Delete a data block from an ECS file (EC:DBLK)

IP1 C: Capability for file (OB.DELBL)
IP2 D: Address of block to be deleted

A block can be deleted from a file as long as it is not referenced by an entry in some subprocess map (reference count = 0). The user must supply the capability index for the file and the address within the file of the block which is to be deleted. If the block is referenced by a map entry, an error is issued.

Possible errors while deleting a data block from an ECS file:

Class	Number	Description
E.PARMS	E.NEGIX	C-list is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPAR	Pointer is negative
E.PARMS	E.BIGPAR	Pointer is too large
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NOBLK	Block to be deleted does not exist
E.FILES	E.INMAPS	Block to be deleted is in a map

4.1.7 Delete a block from a disk file (DF:DBLK)

IP1 C: ECS file capability for locally open disk file (OB.DDLBL)
IP2 D: Starting address in file
IP3 D: Word count

Data blocks may be deleted from a locally open disk file. The file is specified by giving the ECS file capability (IP1) for a locally open disk file. More than one block may be deleted at a time. Blocks are destroyed beginning with the block containing the starting file address (IP2) through the block containing the starting address plus the word count (IP2 + IP3).

Blocks may not be deleted from a file which is open by more than one process. The blocks to be deleted may be either on the disk (unattached) or in ECS. If in ECS, they must be attached only to the calling process. Furthermore, they may not be in any maps.

The disk space occupied by the deleted data blocks is refunded to the disk accounting record associated with the file. Space for any pointer blocks which may become empty is not refunded until the file is globally closed (no longer in use) or pseudo-closed (a technique for updating the disk version of a file). If some of the data blocks were locally attached, swapped ECS space is refunded to the calling program.

The operation to delete disk file data blocks will F-return if the disk file is frozen. This action returns an error on one-level disk files. The data block of a one-level disk file always exists and can never be deleted. An error may occur after one or more blocks have been deleted. The file address of the first block not deleted is added to the error number to facilitate error recovery.

Possible errors while deleting a block from a disk file:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPAR	Negative starting file address or negative word count
E.PARMS	E.BIGPAR	File address plus word count exceeds file length
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NODFIL	No such disk file open by calling process
E.FILES	E.TMOPN	File is open by some other process
E.FILES	E.DIOERR[3]	I/O error reading pointer block (modifier = file address of first block not deleted)
E.FILES	E.NOBLK[3]	Block to be deleted does not exist (modifier = file address of first block which

		does not exist)
E.FILES	E.TMGA[3]	Block is attached to another process (modifier = file address of attached block)
E.FILES	E.INMAPS[3]	Block in some map in calling process (modifier = file address of block in map)
E.FILES	E.ZLEV	Attempt to delete data block of a one-level file

---

[3] Error may occur after zero, one or more blocks have been deleted.

4.1.8 Delete an ECS file (EC:DFIL)

IP1 C: Capability for ECS file (OB.DSTRY)

When an ECS file is deleted, it must not contain any data blocks, i.e., it must consist only of the file descriptor. Only the capability index of the file is required as a parameter.

Possible errors while deleting an ECS file:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full c-list
E.FILES	E.NOFIL	File to be deleted does not exist
E.FILES	E.NOTEMP	File to be deleted is not empty
E.OPER	E.CAPTY	Type or options bad

4.1.9 Delete a Disk File (DF:DFIL)

IP1 C: Capability for a disk file (OB.DSTRY)

The disk file to be deleted must be locally open and it must not be opened by any other process. Blocks may exist in the file and may be attached but no block of the file may be in a map. If everything is well, swapped and fixed ECS are immediately refunded to the local accounting record. The disk space occupied by the file will be refunded and the ECS incarnation of the file is destroyed by a parallel process which will complete the file destruction.

Possible errors in deleting a disk file:

Class	Number	Description
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NODFIL	No such locally open disk file
E.FILES	E.TMOPN	File is open in some other process
E.FILES	E.INMAPS	Block of file is in a map (modifier=file address of offending block).

4.1.10 Move an ECS File Block (EC:MBLK)

IP1 C: Capability for source file (OB.RDFIL, OB.DELBL)
IP2 D: Address in source file of block to be moved
IP3 C: Capability for destination file (OB.WFILE, OB.CREBL)
IP4 D: Address in destination file of block to be moved

File blocks can be transferred between ECS files whose data block sizes (Sn) are equal. In addition to the capabilities for the source and destination files, the system expects to receive from the user the address within the source file of the block to be moved and the address in the destination file to which the block is being moved. Any address within each block suffices. If the block to be moved is referenced by a map, moving it (which deletes it from the source file) would cause problems when swapping; therefore an error is issued. If no errors occur, the designated block is deleted from the source file and added to the destination file. The contents of the block are not affected.

Possible errors while moving a block:

Class	Number	Description
E.FILES	E.NCBLK	Block to be moved does not exist in source file
E.FILES	E.ISBLK	A block already exists at the designated address in the destination file
E.FILES	E.MISMCH	Files do not have equal data block sizes
E.FILES	E.INMAPS	Block to be moved is in a map
E.PARMS	E.NEGPT	File address negative
E.PARMS	E.BIGPT	File address too large
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.OPER	E.CAPTY	Type or options bad

4.1.11 Open a disk file read-only (read/write) (DF:OPRO,DF:OPRW)

Input parameter:
IP1 C: Disk file capability (OB.OPEN (OB.WFILE))

Returned parameter:
RCAP : ECS file capability for open disk file

Before a disk file can be manipulated, it must first be logically opened. The open action returns an ECS file capability for the ECS incarnation of the disk file. Almost all disk file operations require this ECS file capability as a parameter. To facilitate file sharing, a disk file may be opened by several processes and may be opened repeatedly by a single process. A process local open count permits repeated opening and closing by a single process without interferring with the global open count, which reflects the number of separate processes which have logically opened the file.

The disk file capability (IP1) contains the disk address and unique identification for the disk file. The first time a disk file is opened, the header block for the file must be read from the disk and an ECS file created from the shape description contained therein. Subsequent opens return a capability for the ECS file created by the initial open without requiring a disk reference.

If the file has previously been "exclusively" opened, the open action will F-return.

If the file has not been opened previously by the calling process, the process is charged for the fixed ECS space required for the ECS file descriptor and the first level of the file tree. On the disk, the first level of the file is stored in the header block of the file and must be brought to ECS when the file is opened. If the file is frozen or is already open in the calling process, no charge is made for fixed ECS space. Fixed ECS space is funded by the process allocation block; a one-level file must pay for space equal to the data block size plus four; a multi-level file pays for space equal to the number of levels plus the number of branches from the first level (S1) plus four. In addition, sufficient DDS space (disk system global storage) is reserved to allow the attaching of any one file block. Additional DDS space may be reserved explicitly or automatically as needed.

An ECS file capability, with or without write access, is returned depending on whether the open read only or open read/write operation was called for. The "open read/write" operation requires an additional option bit (OB.WFILE) in the disk file capability (IP1) to satisfy the ECS system parameter checking. "Open read/write" returns extra options OB.WFILE, OB.DCRBL, and OB.DDLBL. In all cases, the option bits of the ECS file capability are ANDed with the option bits of the user supplied disk file capability (IP1) before the ECS file capability is returned

to the caller. Finally the following options are turned off in the FCS file capability which is returned: OB.DSTRY, OB.CHNAM, OB.CREBL, OB.DELBL, OB.PLMAP, OB.FDAE, and OB.TRDB.

Prohibited options are:

OB.DSTRY	Destroy a file
OB.CHNAM	Change unique name
OB.CREBL	Create a block
OB.DELBL	Delete a block
OB.FDAE	Direct ECS access
OB.PLMAP	Place in subprocess map
OB.TRDB	Test and reset dirty bit

F-return condition: file is "exclusively" open already.

Possible errors while opening a disk file:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.FILES	E.NOFIL	No such disk file or Disk parity error on header block
E.FILES	E.TMOPN	Too many opens (local open count > $2^{18}-1$ or global open count > $2^{18}-1$)
E.FILES	E.NOLFH	Too many locally open files (> 25)
E.FILES	E.FULL	Disk system tables full
E.OPER	E.CAPTY	Type or options bad
E.ABLOCK	E.NOECS	Insufficient fixed ECS to open file
E.ABLOCK	E.NODDS	Insufficient DDS space

4.1.12 Exclusively open a disk file (DF:XOPN)

Input parameter:

TP1 C: Disk file capability

Returned parameter:

RCAP: ECS file capability for open disk file

In order to provide enforced exclusive access to a disk file to alleviate some problems surrounding those disk system actions which require that the file not be shared (delete file and interchange contents), an exclusive open is provided. This action, if it succeeds, will prevent any subsequent opens on the file. The exclusive open will fail if the file is already exclusively opened or if any other process currently has the file open. If the exclusive open succeeds, it will perform all the activities normally associated with an open of a file file (see "open disk file"). The file is automatically opened read/write.

F-return condition: file already exclusively open or file open by another process.

Possible errors while exclusively opening a disk file:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.FILES	E.NOFIL	No such disk file or Disk parity error on header block
E.FILES	E.FULL	Disk system tables full
E.OPER	E.CAPTY	Type of options bad
E.ABLOCK	E.NOECS	Insufficient fixed ECS to open file
E.ABLOCK	E.NODDS	Insufficient DDS space

4.1.13 Close a disk file (DF:CLO - ECS file capability)
(DF:CLOS - Disk file capability)

IP1 C: ECS file capability (or disk file capability) for locally open disk file (OB.CLOSE)

A disk file should be closed when a process is through manipulating it. If the calling process is the only process to have opened the file, a close will cause the contents of the file on the disk to be updated to reflect all changes in the file content and size. Should the system crash after a successfully completed close, it is unlikely that information in or about the file will be lost. On the other hand, as long as the file remains open by some process, all changes made since the file was first opened may be lost in the event of a crash.

As mentioned above, a process local open count records multiple opens by a single process. If a close decrements the open count to zero, any blocks of the file which are attached to the process are detached, and the swapped ECS space is refunded to the process. Fixed ECS space is also refunded, unless the file was frozen when the process opened it.

If the local open count becomes zero, any local claims on the file are released and the global open count is decremented. When the global open count becomes zero, a request is sent to a special disk system process which will update the contents of the file on the disk. The update is carried on simultaneously with the execution of the process requesting the close file action. The updating of the file on the disk consists of re-writing any pointer blocks which point directly or indirectly to data blocks which have been modified and written to new locations on the disk. After all pointer blocks have been written, the root level pointers are re-written with the file header at their old address on the disk. In this way, the old contents of the file will not be lost if the system should crash before the update procedure is complete.

Possible errors while closing a disk file:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NOOPFIL	No such locally open file
E.FILES	E.INMAPS	Attached block in a map (modifier = file address of block in map)

4.1.14 Attach file block(s) (DF:ATCH)

IP1 C: ECS file capability for a locally open disk file (?)
IP2 D: Starting address in file
IP3 D: Word count

Data blocks of a disk file may be attached by a process. Attaching a data block will cause the block to be held in ECS until the block is detached. If the data block is not already attached, a disk read is initiated to bring the block from the disk. However, the process does not wait for the block to arrive from the disk. Since several processes may share a file and attach the same block, a global attachment count is maintained for disk file data blocks. If the block being attached is already in ECS by virtue of being attached by some other process, no disk read need be initiated. To permit multiple attaches by a single process, a local attachment count is used to prevent one user from over-riding the attaches of another user. Unless the file (IP1) is "frozen", the user is charged for swapped ECS space for each block which was not previously locally attached.

Several data blocks may be attached by one call to the disk system. All data blocks from the block containing the starting file address (IP2) through the data block containing the starting file address plus the word count (IP2 + IP3) are attached. If IP1 is a one level file, only parameter checking is performed, since the first level of a disk file is brought to ECS (i.e., attached) when the file is opened.

Errors may occur in the process of attaching a sequence of data blocks. If the problem is a non-existant block, an F-return is made to the caller. The address of the missing data block is returned in X6. On all other errors, the file address of the block causing the error will be added to the error number. All blocks preceding the error block will already be attached.

Possible errors on attaching file block(s):

Class	Number	Description
E.PARMS	E.NIGIX	C-list index is negative
E.PARMS	E.BIGIX	C- list index exceeds full C-list
E.PARMS	E.NEGPAR	Negative starting file address or negative word count
E.PARMS	E.BIGPAR	Starting file address plus word count too big
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NABR*	No attached block records (disk system out of local storage) (modifier = file address of last block attached)
E.FILES	E.TMA*	Too many local attaches (> 2^{12} - 1) (modifier = file address of last block

		attached)
E.FILES	E.TMGA*	Too many global attaches (> 2^8 - 1)
E.FILES	E.DIOERR*	Disk I/O error (modifier = file address of last block attached)
E.ABLOCK	E.NOSWP	Insufficient swapped ECS in process disk accounting record
E.ABLOCK	E.NODDS	Out of DDS space

---

* May occur after zero, or more blocks are attached.

4.1.15 Detach Disk File Blocks (DF:DTCH)

IP1 C: ECS file capability for a Starting file address
IP2 D: Starting file address locally open disk file (OB.)
IP3 D: Word count

When portions of a disk file are no longer needed in ECS, the corresponding blocks of the file should be detached. In detaching a disk file block, the local attachment count is decremented. If the local attachment count goes to zero, the global attachment count for the block is decremented. If the global count becomes zero, the block is removed from the ECS incarnation of the disk file. If the block is clean (i.e., the dirty bit on the ECS file block is not set), the block does not need to be written to the disk. Otherwise, the block is written to the first available position on the disk. The newly written block will not be linked (on the disk) to its file header until a close or pseudo-close is performed on the file.

Unless the file is frozen, swapped ECS space will be refunded on all blocks for which the local attachment count has become zero. As with "attach", more than one block may be detached by a single call on the disk subsystem. Also, errors may occur after one or more blocks have been detached. In this case, the file address of the offending block is added to the error number.

Possible errors while detaching a file:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPAR	Negative starting file address or Negative word count
E.PARMS	E.BIGPAR	Starting file address plus word count too big
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NATH	Block not locally attached (modifier = file address of block)
E.FILES	E.DIOERR	Disk I/O error on block (modifier = file address of block)
E.FILES	E.TMD	Too many detaches (block is in a map and non-map attach count already zero) (modifier = file address of block)

4.1.16 Interchange file contents ("shazam") (DF:SHAZ)

IP1 C: ECS file capability for primary file (OB.DCRBL, OB.PDLBL)
IP2 C: ECS file capability for secondary file (OB.DCRBL,OB.DDLBL)

It is possible to interchange the contents of two disk files without copying data from one file to the other. Whenever one file is considered to be the "backup" version of another file containing the updated version of the file, the file interchange action may be used to securely update the backup file. The file interchange is performed by first updating the contents of both files on the disk and then interchanging the root pointers of the two files in ECS and on the disk. The order of disk writes is such that the secondary file header is clobbered first. Then the primary file is updated to contain the former contents of the secondary file. Finally, the secondary file header is updated on the disk to point to what was formerly the contents of the primary file. This algorithm guarantees that a system failure at any point will leave the primary file either unchanged or containing the contents of the secondary file. The secondary file will be 1) unchanged, 2) gone, or 3) will contain the previous contents of the primary file. After the interchange, all blocks of both files will be "detached" (i.e., on the disk).

In order to interchange two files, several conditions must be met. Both files must be locally open and must not be opened by any other process. No block of either file may be in a map and the accounting record of the smaller file must fund the increase in file size. The accounting record of the larger file will be refunded an appropriate amount of space. One level files cannot participate in a file interchange.

Possible errors while interchanging file contents:

Class	Number	Description
E.OPER	E.CAPTY	Type or options bad
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.FILES	E.NODFIL	File not locally open disk file (modifier=1 if secondary file; 0 if primary file)
E.FILES	E.INMAPS	Block in a map (modifier=file address of offending block)
E.FILES	E.TMOPN	File is open in another process (modifier=1 if secondary file; 0 if primary file)
E.FILES	E.ZLEV	File is one level file (modifier=1 if secondary file; 0 if primary file)
E.ALLOC	E.NODSK	Insufficient disk space in file accounting record

4.1.17 Read (write) file (EC:READ, EC:WRITE)

IP1 C: Capability for file (OB.RDFIL,(OB.WFILE))
IP2 D: Address in file
IP3 D: Address in Central Memory
IP4 D: Count of words to be transferred

The action of reading (writing) an ECS file transfers words between the address space of the running (current) subprocess and the data blocks of a file. In addition to the capability index for the file, the user specifies the address in the file of (for) the desired information, the address in Central Memory of the area to be read into (written from), and the number of words that are to be read (written). If a transfer is requested which involves a file address corresponding to a non-present data block, the transfer proceeds until the non-present file address is encountered, whereupon F-return action passes control to the disk system. The disk system will check to see if the file (IP1) is a locally open disk file. If not, an F-return will again be initiated (see "Process control" and "Operations"). If the file is a disk file, the missing blocks will be fetched from the disk, the transfer performed, and the blocks released or returned to the disk. If a block which does not exist in the disk version of the file is encountered, an F-return is initiated.

Possible errors while reading (writing) a file:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPAR	Word count negative
E.PARMS	E.NEGPT	File address negative or CM address negative
E.PARMS	E.BIGPAR	File address plus word count (IP2 + IP4) exceeds file length or CM address plus word count (IP3+IP4) exceeds user's field length
E.OPER	E.CAPTY	Type or options bad

4.1.18 Make an exclusive claim on a file (with or without wait)
(DF:ECLM) (with wait)
(DF:ECLF) (without wait)

IP1 C: ECS file capability for a locally open disk file (OB.ECLM)

An exclusive claim may be honored only when no other claim (exclusive or shared) has been issued for the file in question. If the "claim without wait" call is used, an immediate F-return is passed to the user if the claim cannot be satisfied immediately. If the "claim with wait" call is used, the process is added to the end of the claim wait queue for the file (IP1) if the claim cannot be satisfied immediately. This queue is used to keep track of claims which cannot be honored due to outstanding claims on the file. Whenever a claim is released (see below), the claim wait queue is checked. If non-empty, either a single exclusive claim is honored from the head of the queue provided no other claims exist on the file, or a sequence of shared claims is processed from the beginning of the queue. To avoid locking up a user process in the disk system, a time limit is enforced in the claim wait queue. If a claim cannot be satisfied after a wait of one to two minutes in the queue, the user is removed from the queue and an F-return is returned to the caller. Finally, the claim mechanisms permit only one claim per user. Any attempt by a single user to make more than one claim on a single file will cause an error to be returned.

Possible errors while making an exclusive claim:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index negative
E.PARMS	E.BIGIX	C-list index too large
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NOFIL	No such open disk file
E.FILES	E.EXCLAM	Local exclusive claim already
E.FILES	E.SHCLAM	Local shared claim already

4.1.19 Make a shared claim on a file (with or without wait) (DF:SCLM)
(DF:SCLF)

IP1 C: ECS file capability for a locally open disk (OB.SCLM)

If the file is not exclusively claimed by some other process and no process is waiting for an exclusive claim, a shared claim is honored. Note that the shared claim can be honored for several processes and has the effect of preventing an exclusive claim. If the claim cannot be honored, the user is either added to the end of the claim wait queue or passed an immediate F-return, depending on the type of the call (wait or no wait). (See above.)

Possible errors while make a shared claim:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index negative
E.PARMS	E.BIGIX	C-list index too large
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NODFIL	No such open disk file
E.FILES	E.EXCLAM	Local exclusive claim already
E.FILES	E.SHCLAM	Local shared claim already

4.1.20 Release claim on a file (DF:RLSE)

IP1 C: ECS file capability for a locally open disk file (OB.FEL)

If there is a claim (shared or exclusive) on the file by the calling process, the file's claim status is updated to reflect the release of the claim. If the claim was an exclusive claim or the last shared claim on the file, the claim wait queue is processed. Either one exclusive claimer or a sequence of shared claimers (whichever occurs first) are activated from the claim wait queue.

Possible errors while releasing a claim:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index negative
E.PARMS	E.BIGIX	C-list index too large
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NODFIL	No such open disk file
E.FILES	E.NOCLAM	No local claim exists

4.1.21 Read shape of an ECS file (EC:RSHP)

IP1 C: Capability for file
IP2 D: Address of buffer for the shape numbers
IP3 D: Buffer size

The shape of a file is described by a sequence of positive integers (S1,S2,...,Sn), each of which is the number of branches in the file tree at each node of level i ($1 \leq i \leq n$). Each Si ($i > 1$) must be a non-negative power of two. The user can obtain these shape numbers by specifying the index of the capability for the file whose shape he wants to read, and the address and size of a buffer for the shape numbers. The number of levels in the file is placed in the first word of the buffer and the shape numbers (S1,...,Sn) are placed in succeeding words until either the buffer is full or all the shape numbers have been passed.

Possible errors while reading shape:

Class	Number	Description
E.FILES	E.NOFIL	File whose shape is to be read does not exist
E.PARMS	E.NEGIX	C-list index negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPT	Buffer address is negative
E.PARMS	E.NEGPAR	Buffer size ≤ 0
E.PARMS	E.BIGPAR	Buffer address plus size exceeds user's field length
E.OPER	E.CAPTY	Type or options bad

4.1.22 Display disk file status (DF:DSF)

Input parameter

TP1 D: ECS file unique name (left justified)

Returned parameter

RDAT: ECS file unique name, FHR, and LFH (11 words)

The indicated file is located and the ECS file unique name, the file header record, and the local file header are returned to the caller. The format of the returned data is indicated in the figure.

Possible errors while displaying disk file status:

Class	Number	Description
E.FILES	E.NODFIL	No such locally open file
E.		Return parameter with errors

4.1.23 Display status of n-th disk file (DF:DSFN)

Input parameter

TP1 D: Local index of file to display

Returned parameter

RDAT: ECS file unique name, FHR, and LFH (11 words)

The date is returned as with "display disk file status". If the number of locally open files is less than TP1, an F-return is initiated.

Possible errors while displaying status of n-th disk file:

Class	Number	Description
E.PARMS	E.NEGPAR	IP1 $\leq$ 0

4.1.24 Display n-th attached block of disk file

Input parameters

IP1 D: ECS file unique name (left justified)
IP2 D: index of block to display

Returned parameter:

RDAT: Attached block record (2 words)

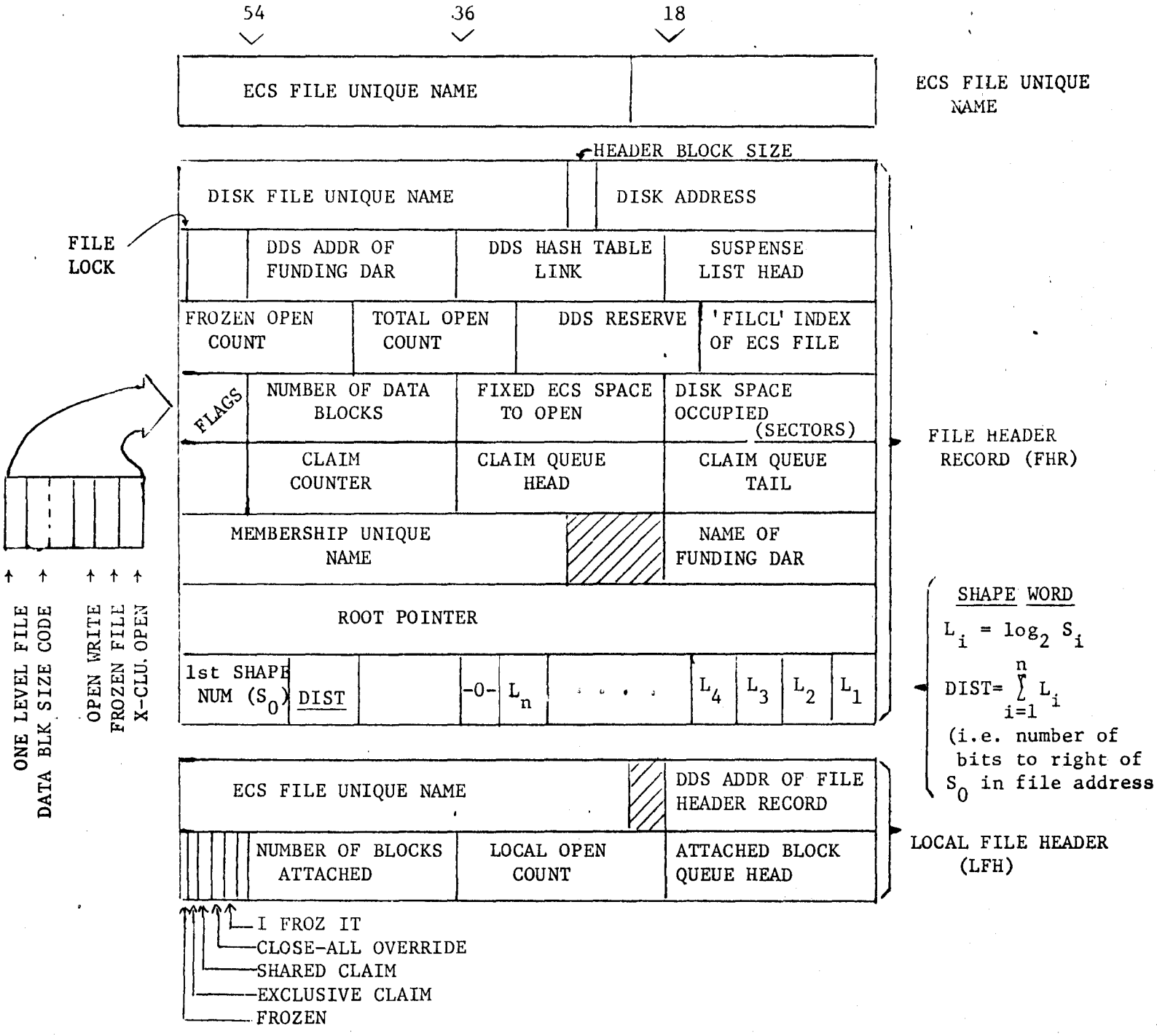
The IP2-th attached block record of the file indicated by IP1 is returned to the caller. If there are fewer attached block records than IP2, an F-return is initiated.

Possible errors while displaying n-th attached block of disk file:

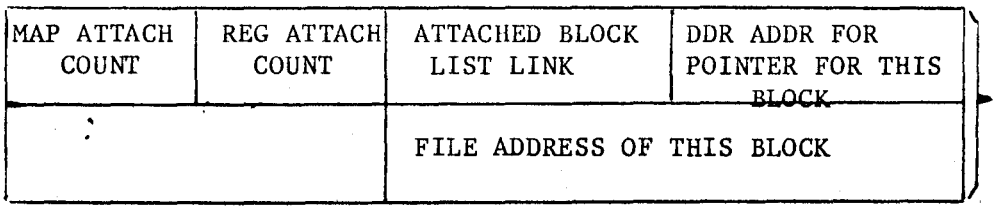
Class	Number	Description
E.FILES	E.NODFIL	No such open disk file
E.NEGPAR	E.PARMS	IP2 $\leq$ 0

DATA RETURNED BY DISPLAY FILE

DF:DFHR



DATA RETURNED BY DISPLAY FILE BLOCK



4.1.25 Check for missing ECS data blocks ("probe") (EC:PROB)

IP1 C: Capability for ECS file
IP2 D: Address of block in the file

This action allows the user to check for the presence of a data block. The parameters required are the index of the capability for the file to which the block belongs, and the address within the file where the block is supposed to be located. The number of missing levels in the path from the root of the file tree to that particular block is returned in register X6. Thus, if the block is present, X6 <- 0; if the n level file is empty, X6 <- n; and if only the data block is missing (its pointer block is present), X6 <- 1.

Possible errors while checking for missing blocks:

Class	Number	Description
E.FILES	E.NOFIL	The file does not exist
E.OPER	E.CAPTY	Type or options bad
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPAR	The address of the block is negative
E.PARMS	E.BIGPAR	The address of the block is too large

4.1.26 Check for missing disk file blocks ("probe") (DF:PROB)

IP1 C: ECS file capability for a locally open disk file
IP2 D: Address of a block in the file

The existence of data blocks and pointer blocks in disk files can be checked with this action. The result is returned in register X6 and is interpreted the same as for the "probe" of an ECS file. The first level of a disk file always exists, and thus the empty n-level disk file would return X6 = n-1 for all file addresses. Disk file "probe" may indicate fewer missing levels than there really are if blocks have been deleted since the last "close" or "pseudo-close" of the file. An I/O error on a data or pointer block is treated as if the block were missing from the file.

Possible errors during a disk file block probe:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.OPER	E.CAPTY	Type or options bad
E.FILES	E.NODFIL	No such locally open file
E.FILES	E.BIGPAR	File address too big
E.FILES	E.NEGPAR	File address negative

4.1.27 Close all open files [5] (DF:CAOF)

No input parameters

This action closes all locally open files which do not have the close-all-over-ride flag set. It is to be used primarily to clean up the user process between major job steps or after things have gotten fouled up. The action will abort if a block is in the map of some file which is being closed.

Possible errors while closing all open files:

Class	Number	Description
E.FILES	E.INMAPS	Some block of some file is in a map (file address added to error number)

---

[5] Privileged operation.

4.1.28 Set/reset close all over-ride [5] (DF:CFLG)

Input parameters:

TP1 C: ECS file capability for an open disk file
IP2 D: Zero for reset; non-zero for set

Returned parameter:

RDAT: (1) old value of over-ride flag

The setting of the flag which over-rides the close-all action is controlled by this action. For a file to remain open through the close-all action, the over-ride flag must be set. If IP2 is zero, the over-ride flag is reset. Otherwise, the over-ride flag is set to prevent the file from being closed by the "close-all open files" action.

Possible errors while setting-resetting close-all over-ride:

Class	Number	Description
E.FILES	E.NODFIL	No such locally open disk file
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.OPER	E.CAPTY	Type or options bad

---

[5] Privileged operation.

4.1.29 Test and reset dirty bit [6] (EC:TRDB)

IP1 C: Capability for file (OB.TRDB)
IP2 D: Any address within block to be tested

Returns 0 in X6 if the block was clean. Returns 1 in X6 if the block was dirty. F-returns if specified block does not exist.

A bit on each data block of a file is used to tell whether or not the block has been written in since it was last tested. A complete description of the logic controlling the bit is

1. Data blocks are created clean.
2. Blocks are dirtied by:
   a. file writes to any part of the block, including writes with a word count of 0;
   b. being put in a map R/W;
   c. being put in a DAE map entry
3. Move block carries the dirty bit along with the block.
4. Test and reset leaves the block clean.

Possible errors while testing and resetting dirty bit:

Class	Number	Description
E.FILES	E.NODFIL	File does not exist
E.PARMS	E.NEGPAR	File address negative
E.PARMS	E.BIGPAR	File address too large
E.PARMS	E.NEGIX	C-list index negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.OPER	E.CAPTY	Type or options bad

---

[6] Privileged operation.

4.1.30 Return disk subsystem clocks (DF:CLKS)

No input parameters

Returned parameters:

RDAT 0: system time
1: swap time
2: disk sys time

The cumulative time expended by the disk subsystem action is returned to the caller.

4.4 C-LIST ACTIONS

Create a C-list
Display a Capability from the Full C-list
Copy a Capability within Full C-list and Decrease the Options
Copy Capability from Full C-list to Arbitrary C-list (and vice-versa)
Change Unique Name
Zero a Capability
Create a Capability Creating Authorization
Create a Capability of Authorized Type
Destroy a C-list

4.4.1 Create a C-list (EC:CCL)

IP1 C: Capability for allocation block (OB.CRECL)
IP2 D: Index in full C-list to return new capability
IP3 D: Length of new C-list

A capability list (C-list) is a sequence of capabilities and "empty" positions. Upon creation each C-list is filled with "empties" (zero words). To create a capability list, the user must supply the index of the Allocation block which funds the space occupied by the C-list (IP1). In addition to the length of the new C-list (IP3), the user must supply an index in the full C-list for the capability for the new C-list (IP2).

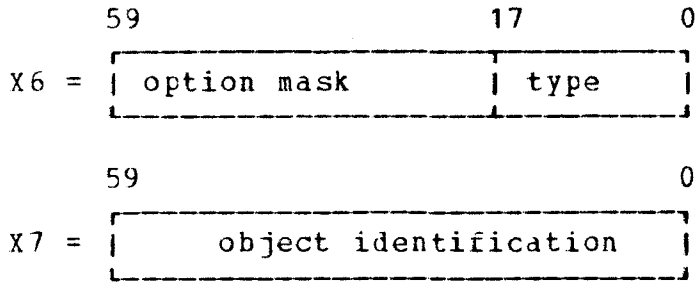
Possible errors while creating a C-list:

Class	Number	Description
E.ABLOCK	E.NOABLK	Allocation block does not exist
E.ABLOCK	E.NOECS	No ECS available
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.PARMS	E.NEGPAR	Length of new C-list ≤ 0
E.PARMS	E.BIGPAR	Length of new C-list exceeds core buffer area

4.4.2 Display a Capability from the Full C-list (EC:DSCP)

IP1 C: Index in full C-list

When referring to capabilities within the full C-list, the capability index used is interpreted as if the C-lists in the full C-list were joined to form one long C-list. Thus, the index of the desired capability (IP1) is all that is required to display it. The two words of the capability are returned in X6 and X7.



The following objects may be specified by the type field:

Object	Type Number
A.K.	2737B[1]
Allocation Block	1767B
CCA	1773B
Class Code	1737B
C-list	1377B
Directory	1776B[1]
Disk File	1775B[1]
ECS file	1577B
Event Channel	1757B
Name tag: Dynamic	2677B[1]
Static	2577B[1]
Operation	1677B
Subsystem	2377B[1]

Possible errors while displaying a capability:

System Entry/Exit Errors only.

---

[1] Subject to change.

4.4.3 Copy a Capability within Full C-list and Decrease the Options (EC:MCAP)

IP1 C: Index of desired capability
IP2 D: Index of destination C-list entry
IP3 D: Mask of options to preserve (in bottom 42 bits - top 18 ignored)

The user can copy a capability from one location in the full C-list to another and in doing so may decrease the number of allowed options. Recall that when an object is created, a capability is returned which has all the option bits (the high order 42 bits of the first word) set. The user must indicate the C-list index of the capability he wishes to copy (IP1), the C-list index where the altered capability will be placed (IP2), and a bit-mask which will first be logically shifted, and then "ANDed" with the option bits of the original capability (IP3) to produce the option mask for the new version of the capability.

Possible errors while copying a C-list and decreasing the options:

Class	Number	Description
E.PARMS	E.NEGIX	Index of desired capability is negative
E.PARMS	E.NEGIX	Index of destination C-list entry is negative
E.PARMS	E.BIGIX	Index of desired capability is too large
E.PARMS	E.BIGIX	Index of destination C-list entry too large

4.4.4 Copy Capability from Full C-list to Arbitrary C-list (and vice-versa (EC:CIN, EC:COUT)

IP1 C: Destination (source) C-list (OB.CPYIN, (OB.CPYOUT))
IP2 D: Index within destination (source) C-list of capability
IP3 D: Index in the full C-list of source (destination) capability

These two actions allow the user to transfer a capability between the full C-list and an arbitrary C-list. Two parameters are required to indicate the location of the capability in the arbitrary C-list, and a third to locate the capability in the full C-list.

Possible errors while copying a capability from a full C-list to an arbitrary C-list:

Class	Number	Description
E.MISCE	E.CLMOT	C-list does not exist
E.PARMS	E.NEGIX	IP2 is negative
E.PARMS	E.NEGIX	IP3 is negative
E.PARMS	E.BIGIX	IP2 is too large
E.PARMS	E.BIGIX	IP3 is too large

4.4.5 Change Unique Name[2] (EC:CHUN)

IP1 D: C-list index of ECS system object (OB.CHNAM)

This action allows the user to change the unique name of an object. The system generates a new capability for the object with all option bits set, thereby invalidating all old capabilities for that object. The capability for the object whose name is to be changed must carry the option bit which allows such a change (OB.CHNAM). If the object is a file for which there are references in any map entries, all such maps will be recompiled.

Possible errors while changing unique name:

Class	Number	Description
E.MISLE	E.MISSOB	No such object

4.4.6 Zero a Capability (EC:ZCAP)

IP1 D: Index in full C-list of the capability

This action erases a capability by storing zeros in the indicated capability slot in the full C-list.

Possible errors while zeroing a capability:

Class	Number	Description
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list length
E.PARMS	E.NOTCL	Not a C-list capability (indirection)
E.MISLE	E.CLMOT	C-list does not exist (indirection)

---

[2] Privileged operation.

4.4.7 Create a Capability Creating Authorization (EC:MCCA)

IP1 D: C-list index for returned authorization

A capability creating authorization is a special type of capability. Such a capability may be used to create new capabilities. The second word of the capability contains the type of capability which may be manufactured under the authorization.

Possible errors while creating an authorization:

Class	Number	Description
E.MISCE	E.NOAUTH	No more capability types are available [3]

4.4.8 Create a Capability of Authorized Type (EC:CCAP)

IP1 D: C-list index for returned capability
IP2 C: A capability creating authorization
IP3 D: Data for second word of returned capability

A capability of the type specified by IP2, with all option bits on, and with second word equal to IP3, is returned at the specified index in the caller's C-list.

Only the entry/exit errors are possible.

---

[3] Note that since there are only about 48000 different capability types, unrestricted use of this operation would allow one user to exhaust the supply, thus making those that wanted a special capability type later on very unhappy.

4.4.9 Destroy a C-list (EC:DCL)

IP1 C: Capability for C-list (OB.DSTRY)

The user may destroy a C-list when he no longer needs it; only the index of a capability for the C-list is required. If the C-list to be destroyed is in the full path of the user's process, an F-return is initiated and the C-list is not destroyed.

Possible errors while destroying a C-list:

Class	Number	Description
E.MISCE	E.CLMOT	C-list does not exist

4.6 EVENT CHANNEL ACTIONS

Create an Event Channel
Send an Event
Get an Event or Hang
Get an Event or F-return
Get an Event or Hang (Multiple)
Get an Event or F-return (Multiple)
Destroy an Event Channel

4.6.1 Create an Event Channel (EC:CEVC)

IP1 C: Capability for allocation block (OB.CREEC)
IP2 D: C-list index for new event channel capability
IP3 D: Number of events that queue can hold

When an event channel is created it consists of a three word header and an event queue which is initially empty. The header words are used to maintain the queue of events and a queue of waiting processes, which develops if the queue of events becomes empty and processes request events from that channel. When creating an event channel, the user specifies a capability for an allocation block (IP1) which funds the ECS space occupied by the event channel, a C-list index (IP2) where the system can put the capability (with all options allowed) for the event channel when it creates it, and the length (number of possible events) of the event queue (IP3).

Possible errors while creating an event channel:

Class	Number	Description
E.ABLOCK	E.NOABLK	Allocation block does not exist
E.ABLOCK	E.NOECS	No ECS available
E.PARMS	E.NEGIX	C-list is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.EVENT	E.NEGQ	Length of event queue ≤ 0
E.EVENT	E.BIGQ	Event queue too large
E.OPER	E.CAPTY	Type or options bad

4.6.2 Send an Event (EC:SEV)

IP1 C: Capability for the event channel (OB.SNDEV)
IP2 D: Datum part of event

This action allows the user to send an event to an event channel. The user specifies the capability for the event channel (IP1) and a 60-bit datum to be passed with the event (IP2). The system indicates the disposition of the event to the user in X6. The following responses are possible:

Condition	Response
Event put in event queue	1
Event passed to a process	2
"YOU LOSE" event put in event queue	3
Event queue full	4

The first response indicates that all went well, and there was no process awaiting an event in the process queue. The second response indicates that there was a process waiting in the queue and that it was passed the event. The third response indicates that there was only one free slot in the event queue (an event occupies two words); the intended datum has been replaced by a "you lose" datum (-0) so that the process which ultimately gets the event will be aware that the event queue was full and that information was lost. The fourth response indicates that no action was taken because the queue was full.

Possible errors resulting from sending an event:

Class	Number	Description
E.EVENT	E.NOCHAN	Event channel does not exist
E.OPER	E.CAPTY	Type or options bad
E.PARMS	E.NEGIX	C-list is negative
E.PARMS	E.BIGIX	C-list exceeds full C-list

4.6.3 Get an Event or Hang (EC:GEVH)

IP1 C: Capability for event channel (OB.GETEV)

A user requests an event from a channel by passing the C-list index of the capability of the channel in question (IP1). If the event queue is empty, the process must wait ("hang" or "block") until an event arrives before it can resume execution. If more than one process is awaiting an event, the first event sent to that channel is passed to the first process, while the other process(es) continues to wait. The event is returned to the calling process in X6 and X7. X6 contains the unique name of the process which sent the event while X7 contains the event datum. A chaining word index of 1 is packed into X6 (useful mainly in multiple event channel work, see 4.6.5).

Possible errors while getting an event:

Class	Number	Description
E.EVENT	E.NOCHAN	Event channel does not exist
E.OPER	E.CAPTY	Type or options bad
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list

4.6.4 Get an Event or F-return (EC:GEVF)

IP1 C: Capability for event channel (OB.GTEVF)

The user requests an event from a channel using the C-list index of the event channel's capability (IP1). If the event queue is empty, an F-return will be initiated in order to permit the process to take alternative action. The event is returned in X6 and X7 as in 4.6.3 above.

Possible errors while getting an event:

Class	Number	Description
E.EVENT	E.NOCHAN	Event channel does not exist
E.OPER	E.CAPTY	Type or options bad
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list

4.6.5 Get an Event from One of a Set of Event Channels or Hang

(EC:GVMH)

IP1 D: Pointer to list of C-list indices for event channels (OB.GETEV...)
IP2 D: Number of event channels involved

The procedure for getting an event from one of a set of event channels is similar to that for getting a single event (see 4.6.3 above). The channels are interrogated one at a time and if their respective event queue is empty, the user's process will be queued on the process queue of the event channel. If an event subsequently arrives or is discovered on one of the event channels in the list, the process is removed from all the process queues on which it has already been chained and it is passed the event. If no event arrives or is discovered before the last event channel is interrogated, the process must wait ("hang" or "block") until an event arrives on one of the event channels.

When an event is finally passed in X6 and X7, the ordinal in the user's list of the event channel producing the event is packed as the scale of X6 (i.e., if the event came from the first channel in the list, 1 is packed in the scale).

Possible errors while getting an event from a list of channels:

Class	Number	Description
E.EVENT	E.NOCHAN	Event channel does not exist
E.PARMS	E.NEGPAR	Number of channels is ≤ 0
E.PARMS	E.BIGPAR	Pointer to list + number of channels exceeds FL or number of channels exceeds scratch area (P.PARAML - 2 cells)
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.OPER	E.CAPTY	Capability type or options bad

4.6.6 Get an Event from One of Set of Event Channels or F-return

(EC:GVMF)

IP1 D: Pointer to list of C-list indices for event channels (OB.GTEVF...)

IP2 D: Number of event channels involved

This action is similar to the previous one except that if all of the event queues are empty for the event channels specified by IP1, an F-return is initiated in order to permit the process to take alternative action.

When an event is finally passed in X6 and X7, the ordinal in the user's list of the event channel producing the event is packed as the scale of X6; (i.e., if the event came from the first channel in the list, 1 is packed in the scale).

Possible errors while getting an event from a list of channels:

Class	Number	Description
E.EVENT	E.NOCHAN	Event channel does not exist
E.PARMS	E.NEGPAR	Number of channels if ≤ 0
E.PARMS	E.BIGPAR	Pointer to list + number of channels > FL or number of channels exceeds process scatch area (P.PARML - cells)
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-iist
E.OPER	E.CAPTY	Capability type or options bad.

4.6.7 Destroy an Event Channel (EC:DEVC)

IP1 C: Capability for event channel (OB.DSTRY)

An event channel can be destroyed. The only parameter required is the capability for the event channel which is to be destroyed. If there are any processes waiting on the event channel's process queue, an F-return is initiated leaving the event channel intact.

Possible errors while destroying an event channel:

Class	Number	Description
E.EVENT	E.NOCHAN	Event channel does not exist
E.PARMS	E.NEGIX	C-list index is negative
E.PARMS	E.BIGIX	C-list index exceeds full C-list
E.OPER	E.CAPTY	Type or options bad

Appendix A

Table 1. ASCII - Printer Character Mapping

ASCII Char	Printer Graphic	TSS ASCII Code	ASCII Char	Printer Graphic	TSS ASCII Code	ASCII Char	Printer Graphic	TSS ASCII Code
blank	blank	0	@	¬	40	'	≠	100
!	≤	1	A	A	41	a	A	101
"	↓	2	B	B	42	b	B	102
#	≥	3	C	C	43	c	C	103
$	∨	4	D	D	44	d	D	104
%	∧	5	E	E	45	e	E	105
&	∧	6	F	F	46	f	F	106
'	≠	7	G	G	47	g	G	107
(	(	10	H	H	50	h	H	110
)	)	11	I	I	51	i	I	111
*	*	12	J	J	52	j	J	112
+	+	13	K	K	53	k	K	113
,	,	14	L	L	54	l	L	114
-	-	15	M	M	55	m	M	115
.	.	16	N	N	56	n	N	116
/	/	17	O	O	57	o	O	117
0	0	20	P	P	60	p	P	120
1	1	21	Q	Q	61	q	Q	121
2	2	22	R	R	62	r	R	122
3	3	23	S	S	63	s	S	123
4	4	24	T	T	64	t	T	124
5	5	25	U	U	65	u	U	125
6	6	26	V	V	66	v	V	126
7	7	27	W	W	67	w	W	127
8	8	30	X	X	70	x	X	130
9	9	31	Y	Y	71	y	Y	131
:	:	32	Z	Z	72	z	Z	132
;	;	33	[	[	73	{	(	133
<	<	34	\	∨	74	|	blank	134
=	=	35	]	]	75	}	)	135
>	>	36	^	↑	76	~	blank	136
?	≥	37	_	->	77	rubout	blank	137

Table 2
Non-Graphic TTY Character Representation

Character	Internal ASCII Representation	Key Combination Systext Representation	Function
NUL	140	%@	
SOH	141	%A	
STX	142	%B	
ETX	143	%C	
EOT	144	%D	
ENQ	145	%E	
ACK	146	%F	
BEL	147	%G	Bell
BS	150	%H	Backspace
HT	151	%I	Horizontal Tab
LF	152	%J	Line Feed
VT	153	%K	Vertical Tab
FF	154	%L	Page Eject
CR	155	%M	
SO	156	%N	
SI	157	%O	
DLE	160	%P	
DC1	161	%Q	
DC2	162	%R	
DC3	163	%S	
DC4	164	%T	
NAK	165	%U	
SYN	166	%V	
ETB	167	%W	
CAN	170	%X	Delete Line
EM	171	%Y	
SUB	172	%Z	
ESC	173	%[	
FS	174	%	
GS	175	%]	
RS	176	%1	
US	177	%<-	

Appendix B

Error Classes and Numbers

Class	Number		Description
0 E.CHIP			SCOPE Call
1 E.ARITH			Arith Error
2 E.PARMS			Parameter or Pointer Errors
	0	E.NEGPAR	Parameter too small
	1	E.BIGPAR	Parameter too large (Param number is masked into errnum)
	2	E.NEGPT	Pointer is negative
	3	E.BIGPT	Pointer is too large (Pointer is masked into errnum)
	4	E.NEGIX	C-list index is negative
	5	E.BIGIX	C-list index is too large (Index is masked into errnum)
3 E.FILES			File-processing Errors
	0	E.NOFILES	File does not exist
	1	E.ISBLK	Block to be created exists
	2	E.INMAPS	Block is in map
	3	E.NOBLK	Block to be moved does not exist
	4	E.MISMCH	Block sizes not equal for move
	5	E.NOBKD	Block to be destroyed does not exist

6	E.NOTEMP	File to be destroyed is nonempty
7	E.NEGSIZ	Negative shape number
10	E.BIGSIZ	Shape number is too large
11	E.NOTPOW	Shape number is not power of two
12	E.BIGFIL	File size is too great
13	E.IDERR	ECS I/O Error
.		
.		
.		
24	E.LLEV	Toc many levels
25	E.NODFIL	No such open disk file
26	E.NABR	No attach block record
27	E.DIOERR	Disk I/O error
30	E.TMA	Too many attaches
31	E.TMD	Toc many detaches
32	E.NATH	Block not attached
33	E.ZLEV	One level file
34	E.TMOPN	Too many opens (local or global)
35	E.EXCLAM	Already exclusive claim
36	E.SHCLAM	Already local shared claim
37	E.CLOCK	Claim queue lock up (time out)
40	E.NOCLAM	No local claim on release
41	E.NOLFH	No local file header space
42	E.TMGA	Too many global attaches

4	E.SUBP		Subprocess creation, call, and return errors
	0	E.SAMNA	Duplicate subp name
	1	E.NOFATH	Named father does not exist
	2	E.NOBLOC	Block in swapping directive missing
	3	E.COMP	Not enough room for map
	4	E.MACSZ	Process becomes too big
	5	E.NOFIND	Named subp does not exist
	6	E.FULSTK	No room for subp in stack
	7	E.ROOM	No rocm for parameters
	10	E.NCAP	Too many capability params
	11	E.ESTK	Empty stack (on return)
	12	E.STK	Empty stack (on F-return)
	13	E.NLEAF	Attempt to delete subp at root or not leaf of subp tree
	14	E.IFRET	Illegal F-return
	15	E.NOXJ	No CEJ where expected
	16	E.NSTK	Attempt to delete subp in stack
5	E.PROC		Process Creation Errors
	0	E.BLMISS	Block missing in swapping directive
	1	E.NOROOM	Not enough room for map
	3	E.PGONE	Process gone from MOT
6	E.ABLK		Allocation Block Errors
	0	E.NOABLK	Allocation block gone

1	E.NOECS	Not enough space to create object
2	E.NOSLOT	No MOT slot to create object
3	E.NOSWP	No swapped ECS space
4	E.NODSK	No disk space
5	E.NORES	Not enough reserved space for donation
6	E.NOCP	Not enough CP time for donation
7	E.NOMOT	Not enough MOT slots for donation
10	E.NORLC	Not enough reserved space to cover duplication of object during reallocation.
11	E.FATSON	One allocation block not father of other
12	E.CRGER	Resulting charge rate would be illegal
24	E.BADSN	Bad Service Number
25	E.NODDS	Out of DDS records
26	E.BUSY	Cannot destroy accounting block
27	E.RESV	Accounting block holding reserved space
28	E.ACTIV	Accounting block already active
30	E.NOFUND	No fixed ECS allocation block supplied to disk system

7	E.OPER	Operation Interpretation Errors
0	E.IPO	IPO not capability for operation

	1	E.NOOP	Operation not in MOT
	2	E.CAPTY	Capability type or options bad
	3	E.PSANY	Param spec (any) encountered
	4	E.NOTANY	Param spec (any) not encountered
	5	E.USER	Should be user supplied parameter
	6	E.BIGORD	Order too big for scratch area
	7	E.MANPAR	Too many parameters
	10	E.BIGCNT	Block param exceeds count in param spec in operation
8	E.MISCE		Miscellaneous Errors
	0	E.CLMOT	Capability list not in MOT
	1	E.MISSOB	Misc object not in MOT
	2	E.NOAUTH	No more capability creating authorizations are available
9	E.EVENT		Event Channel Errors
	0	E.NEGQ	Event queue too short
	1	E.BIGQ	Event queue too long
	2	E.NOCHAN	Event channel not in MOT
10	E.NOERR		No subp to take error
	0	E.NOERR1	
11	E.MAPS		Error Class for Maps
	0	E.ISDAE	Attempt to change or zero DAE
	1	E.NT1BLK	DAE attempts to bridge blocks

2	E.NOTDAE	DAE action applied to swapping dir
3	E.BADNNS	Bad word count or missing file
4	E.PRENT	Previous entry during make map
5	E.WRGFL	Wrong cap for previous file
6	E.WRFIL	Wrong file on delete map entry

12 E.PANIC — Panics (Interrupts)

0	E.MLDP	Mild panic (ZR test used by some code)
1	E.MJRP	Major panic (NZ test used by some code)

13 EC.DIRECT — Directory Errors

0	EN.BADNM	Bad name given
1	EN.IMPOT	Access-key not in access-list
2	EN.NTOWN	Wrong actions used to delete link entry
3	EN.DUPNM	Cannot have duplicate names
4	EN.NOSPC	Directory is full
5	EN.LOOP1	Softlink chain too long
6	EN.LOOP2	Successor link chain too long
7	EN.NTDSK	Only disk system objects can be hardlinked
10	EN.ISOWN	Wrong action to delete ownership entry
11	EN.BGOPT	More than 42 options given to 'add

		pair'
	12 EN.ISLCK	Duplicate lock for new access pair
	13 EN.NTLCK	No such lock for access pair to be deleted
	14 EN.NTDIR	Even scanlist entries must be directories
	15 EN.NTKEY	Odd scanlist entries must be access-keys
	16 EN.OWNS	Directory to be deleted owns something
	17 EN.RSRV	Directory is currently reserved (ex. open)
14 EC.DYNTG		Dynamic Name Tag Errors
	0 EN.FLOL	Full local open list
	1 EN.BLOC	Too many local opns
	2 EN.FDNT	Full global table
	3 EN.BGOC	Too many global opens
	4 EN.NTOP	No such locally open tag
15 EC.BEADS		Beads Errors

Appendix C - Option Bit Assignments

Object	Mnemonic	Description	Relative Bit Position
Allocation Block	CB.DSTRY	Destroy Allocation Block	0
	CB.CHNAM	Change Unique Name	1
	CB.CREAB	Create Allocation Block	2
	CB.CRECL	Create a C-list	3
	CB.CREFIL	Create a file	4
	CB.CREPR	Create a process	5
	CB.CRESP	Create a subprocess	6
	CB.CREEC	Create an event channel	7
	CB.ALCRD	Create an operation	8
	CB.GIVE	Reserved space donor	9
	CB.GET	Reserved space donee	10
	CB.GCD	Create capability for nth object	11
	CB.INCHR	Increment charge field	12
	CB.GIVCP	CP time donor	13
	CB.GETCP	CP time donee	14
	CB.GIVMT	MCT slot donor	15
	CB.GETMT	MCT slot donee	16
	CB.INMTR	Increment DTS field	17
C-list	CB.DSTRY	Destroy C-list	0
	CB.CHNAM	Change unique name	1
	CB.CPYIN	Copy capability into C-list	2
	CB.CPYOT	Copy capability out of C-list	3
	CB.LOCCL	Local C-list for subprocess	4
File	CB.DSTRY	Destroy a file	0
	CB.CHNAM	Change aunique name	1
	CB.CREBL	Create a block	2
	CB.DELBL	Delete a block	3
	CB.RDFIL	Read a file	4
	CB.WFILE	Write on the file	5
	CB.PLMAP	Place portion of file in map	6
	CB.RDAE	Direct ECS Access	7
	CB.OPEN	(disk file) Open file	8
	CB.CLOSE	(disk file) Close disk file	9
	CB.DCRBL	(disk file) Create block	10
	CB.DDLBL	(disk file) Delete block	11
	CB.ATCH	(disk file) Attach block	12
	CB.DTCH	(disk file) Detach block	13
	CB.DMAP	(disk file) Put in map	14
	CB.RCLM	(disk file) Exclusive claim	15
	CB.SCLM	(disk file) Shared claim	16
	CB.REL	(disk file) Release claim	17
	CB.FREZ	(disk file) Freeze file	18

	CB.TFDB	Test and reset dirty bit	19
	CB.XCPN	(disk file) Exclusive open	20
Process	CB.DSTRY	Destroy a process	0
	CB.CHNAM	Change unique name	1
	CB.SDINT	Interrupted process	2
Subprocess	CB.DSTRY	Destroy subprocess	0
	CB.TEMP	Set temporary part of class code	1
	CB.FATHR	Father subprocess	2
	CB.SPRET	Subprocess may be jump returned to	3
	CB.PCNT	P-counter of subprocess may be modified	4
	CB.INTSP	Interrupt subprocess	5
	CB.CALOP	Subprocess called by operator	6
	CB.SCNSP	Scn subprocess	7
	CB.CHMAP	Create, zero, or change map entry	8
	CB.DAE	Direct ECS Access map entry	9
	CB.STESM	Set Error Selection Mask	10
Event Channel	CB.DSTRY	Destroy event channel	0
	CB.CHNAM	Change unique name	1
	CB.SNDEV	Send an event	2
	CB.GETEV	Get an event (or hang)	3
	CB.GTEVF	Get an event (or F-return)	4
OPERATION	CB.DSTRY	Destroy an operation	0
	CB.CHNAM	Change unique name	1
	CB.ADDOR	Order may be added to operation	2
	CB.CHTYP	Change parameter specification type in an operation	3
	].CHOPT	Change option bits for "user-supplied capability"	4
	CB.CYCP	Copy an operator	5

Directories (to be supplied)

Appendix D. C-list Type Field Values

Object	Type Number
Access Key	2737B[1]
Allocation Block	1767B
CCA	1773B
Class Code	1737B
C-list	1377B
Directory	1776B[1]
Disk File	1775B[1]
ECS file	1577B
Event Channel	1757B
Name tag: Dynamic	2677B[1]
Static	2577B[1]
Operation	1677B
Subsystem	2377B[1]