SPERRY UNIVAC

1100 Series

Executive System

Volume 3

System Processors

For EXEC Level 35R1

Programmer Reference

SPERRY UNIVAC COMPUTER SYSTEMS

UP-4144.31

This document contains the latest information available at the time of publication. However, Sperry Univac reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Sperry Univac representative.

Sperry Univac is a division of Sperry Rand Corporation.

The software levels reflected in this manual for the various system processors are listed below. The software level shown in parenthesis for each processor is the level documented in the previous version of the manual (UP-4144.3).

Collector (MAP)	Level 29R1	(27)
DATA	Level 8R1	(7)
ED	Level 15R2	(14.02)
ELT	Level 7	(7)
FURPUR	Level 27R2	(26)
PDP	Level 12R1	(10)
PMD	Level 32R1	(30.1)
SECURE	Level 20R1	(18)
SSG	Level 17R1	(16)

PRINTED IN U.S.A.

Page Status Summary

Issue: UP-4144.31 Update A

Section	Pages	Update
Gover/Disclaimer		
PSS	1	A
Preface	1, 2	A
Contents	1 thru 3 4 5 6, 7 8 9	A Orig A Orig A Orig
1	1 thru 9 10 11 thru 14	Orig A Orig
2	1 thru 3 4 thru 16 17, 18 19 thru 22 22a 23, 24 25 thru 28 28a 29 thru 41 42, 43 44 thru 61 62 thru 64	Orig A Orig A A Orig A A Orig A Orig A
3	1 thru 31	Orig
4	1 thru 35	Orig
5	1 thru 4	Orig
6	1 thru 3	Orig
7	1 thru 36	Orig
8	1 thru 4	Orig
9	1, 2 3 thru 10	Orig A
9 (Cont.)	10a 11 thru 22	A Orig
10	1 thru 67	Orig
11	1 thru 21	Orig
Appendix A	1 thru 12	Orig
User Comment Sheet		
Total: 330 pages and cover		

Technical changes are indicated by a vertical bar (|) in the outer margin of the updated pages.

Preface

The SPERRY UNIVAC 1100 Series Executive System Programmer Reference manual has been divided into four volumes. These volumes are titled as follows:

1. SPERRY UNIVAC 1100 Series Executive System, Volume 1, Index, Programmer Reference.

   Volume 1 references terms and subjects covered in the other three volumes.

   a. UP-4144.1 provides a consolidated index for UP-4144.2, UP-4144.3, and UP-4144.4, the three volumes intended for use with EXEC level 32R1 and associated system processors and system utility programs.

   b. UP-4144.11 provides a consolidated index for UP-4144.21, UP-4144.31, and UP-4144.41, the three volumes intended for use with EXEC level 33R1 and associated system processors and system utility program.

   c. The Index for the three volumes intended for use with EXEC level 35R1 will be included in each of the three volumes and released as an update package for each volume. It will not be a consolidated index.

2. SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC, Programmer Reference.

   Volume 2 describes the overall control of SPERRY UNIVAC 1100 Series Systems by the Executive System.

   a. UP-4144.2 is the programmer reference for EXEC level 32R1.

   b. UP-4144.21 is the programmer reference for EXEC level 33R1.

   c. UP-4144.21-A provides corrections for UP-4144.21.

   d. UP-4144.21-B updates UP-4144.21 to correspond to EXEC level 33R2.

   e. UP-4144.22 is the programmer reference for EXEC level 35R1.

3. SPERRY UNIVAC 1100 Series Executive System, Volume 3, System Processors, Programmer Reference.

   Volume 3 describes the basic system processors.

   a. UP-4144.3 describes the system processors associated with EXEC level 32R1.

   b. UP-4144.31 describes the system processors associated with EXEC level 33R1.

   c. The addition of Update Package A (UP-4144.31-A) to UP-4144.31 produces a manual which describes the system processors associated with EXEC level 35R1.

4. SPERRY UNIVAC 1100 Series Executive System, Volume 4, System Utility Programs, Programmer Reference.

   Volume 4 describes the System Relocatable Library, system common banks, and utility processors.

   a. UP-4144.4 describes the system utility programs associated with EXEC level 32R1.

   b. UP-4144.41 describes the system utility programs associated with EXEC levels 33R1 and 35R1.

Cross references to subjects in other volumes are by volume, number, dash subsection number, e.g., 2-3.7.4 references volume 2, subsection 3.7.4.

Contents

FIGURES

TABLES

1. Introduction

1.1. SCOPE OF MANUAL

The SPERRY UNIVAC 1100 Series Operating System comprises the Sperry Univac-supplied software for the SPERRY UNIVAC 1100 Series Computer Systems. This volume and Volumes 2 and 4 discuss the base portion of the operating system; that is, the SPERRY UNIVAC 1100 Series Executive System (EXEC 8) and the associated software needed to construct, execute, and maintain user programs.

Information that is primarily of interest only to an operator, installation manager, or systems analyst is described only briefly if at all (for example, operating procedures, system generation procedures, internal system logic, and so forth). Such material is covered in other Sperry Univac publications.

The purpose of this manual is to provide information for the user programmer so that full use of the wide range of capabilities provided by the SPERRY UNIVAC 1100 Series Executive can be made. Any differences between the operating system described in this manual and the latest released software are described in the Software Release Documentation that accompanies each release.

A basic knowledge of the SPERRY UNIVAC 1100 Series system architecture is assumed. For some relatively specialized topics a knowledge of a 1100 assembly language programming may be helpful. However, this is not required for full use of this manual by the users of higher level languages.

This is Volume 3 of the four-volume SPERRY UNIVAC 1100 Series Executive System Programmer Reference. To use this volume it is assumed the reader is familiar with Volumes 2 and 4.

Volume 1 contains the index for all volumes. Volume 2 describes the basic Executive (EXEC), which includes the following:

- Concepts and Definitions
- Executive Control Statements
- Executive Service Requests (ER)
- Symbiont Interface Requests
- Input/Output Device Interfaces
- File Control

- Demand Processing
- Communications Handler
- Real-Time Processing
- Checkpoint/Restart
- Internal Executive Design

Volume 4 describes the following:

- Flow Analysis Program (FLAP)
- System Relocatable Library and System Common Banks
- Document Processor (DOC)
- SNOOPY
- CULL Processor
- LIST Processor

This volume describes the SPERRY UNIVAC 1100 Series System Processors. These are general system programs which are used to construct and modify programs, maintain and modify files, and provide diagnostic information upon program termination. System processors are a logical extension of the Executive system. They normally reside in the file SYS$*LIB$.

In addition to the System Processors, file formats and file maintenance software, which are normally transparent to the user, are discussed. This information is provided to:

- give insight into the file structure used by the FURPUR processor, the language and system processors, and the symbiont complex and;
- enable the user to write application software to build, insert, and retrieve data from files.

1.2. MODIFYING SYMBOLIC ELEMENTS

This information applies only to processors which obtain their input from the System Relocatable Library processor interface routine SIR$, (see Volume 4-2.1.4). The source input/output routine is used by a processor to obtain the source language images from the runstream or from a symbolic element in a program or element file (see 2.2.6) or a SDF file. The routine can automatically merge corrections, list the corrections, and produce an updated symbolic element which is inserted into a program file. The symbolic element which contains the source input may be cycled; the desired cycle is specified in the processor control statement. The source input routine automatically passes to the processor only those images that pertain to the cycle requested.

1.2.1. Line Correction Statement

The lines of text in a file may be considered to be numbered sequentially starting with line one and incrementing by one. When altering the symbolic element, these numbers are used on the line correction statement to indicate where the correction lines are to be inserted. The format of the line correction statement is:

```
-n,m
```

The minus sign in column one is the correction indicator which specifies that symbolic lines n through m are to be replaced by correction lines. The lines immediately following the -n,m construction are inserted until another correction statement is read. If no lines follow the -n,m correction statement, lines n through m are deleted.

The construction -n with the correction indicator appearing in column one specifies that the succeeding lines are to be inserted in the symbolic element after line n.

If correction lines are to be inserted before the first line in the symbolic element, the correction lines are placed immediately after the processor control statement without specifying any insertion line numbers, or by using -0 (negative zero) as the line indicator.

The terms in succeeding correction statements must form a non-decreasing sequence since the corrections are applied by means of a one-pass merge of the symbolic file and the corrections. Thus, if -n1,m1 follows -n0,m0, n1 must be greater than m0. If -n1,m1 follows -n0, n1 must be greater than n0. If -n1 follows -n0,m0, n1 must be greater than or equal to m0. The exclamation point (!) is used to indicate the last line in the input element. -m,! would delete all lines of the element from line m through the last line and allow insertions after the last line. -! would allow line insertion after the last line of the element. -!,! is not allowed.

Examples:

```
@ASM,U        DF3.WINDUP,.WINDUP
-30,31
CORRECTION LINE A
-100,115
-120
CORRECTION LINE B
CORRECTION LINE C
CORRECTION LINE D
-150,150
@MAP,IL
```

The U option on the @ASM control statement specifies that the next higher cycle of symbolic element WINDUP is to be produced. Lines 30 and 31 are replaced by correction line A. Lines 100 through 115 are to be deleted. Correction lines B, C, and D are inserted after line 120. Line 150 is deleted. Encountering the @MAP control statement indicates that there are no more correction lines to the symbolic element.

1.2.2. Redefinition of the Correction Indicator

It is possible to redefine the correction indicator so that a symbol other than the minus sign may indicate the insertion of correction lines. Redefinition of the correction indicator makes it possible to insert correction lines which contain a minus sign in column one. The format for redefinition is:

```
-=x
```

x may be from one to three nonblank characters in length. The correction indicator may be redefined any number of times at any position within the correction stream but only one symbol is recognized as the correction indicator at any time. If x is blank, the statement is ignored.

Examples:

```
1.  @DATA          FILE1,FILE2
2.  -2
3.  CORRECTION LINES
4.  -=*
5.  *11,13
6.  CORRECTION LINE A
7.  CORRECTION LINE B
8.  *=+++
9.  +++22
10. CORRECTION LINE
11. @END
```

Line 2 indicates correction lines are to follow line 2 in the source program. Line 4 redefines the correction indicator to an asterisk (*). Line 5 indicates lines 11, 12, and 13 are replaced with correction lines. Line 8 redefines the identifier to +++. Line 9 indicates correction line follows line 22 of the source program.

1.2.3. Partial Line Corrections

In addition to inserting entire symbolic correction lines, partial line corrections are also permitted. This is accomplished by using a range correction statement to define the number of code lines to be partially corrected followed by change correction statements which define the correction to be made.

In the formats given in 1.2.5 and 1.2.6 the slash (/) is used as a separator character. The separator character may be any character other than a digit, a comma if the change statement is Format 1, a blank, or a correction indicator for Format 3 or 4 (see 1.2.5). The first separator may be preceded by any number of blanks. The character chosen as a separator must not appear as a character in the old-data and new-data parameters of the change correction statement.

1.2.4. Range Correction Statement

The range correction statement formats are:

Format 1: -x,y-

Format 2: -x-

The minus character immediately following the y is an edit indicator which must always be coded as a minus character. (The minus character that immediately precedes the x is a correction indicator

that may be redefined by the user, see 1.2.2). The range correction statement must be followed by one or more change correction statements and there must be one change correction statement for each statement in the range –x,y–. For example, if the range correction statement is:

--2,7–

then there must be six change correction statements immediately following the range correction statement. If the number of change correction statements following the range correction statement does not equal the number given on the range correction statement, a diagnostic is given.

If format 2 is used, the following corrections are all applied to line x. The first correction is to line x and the following corrections are to the last corrected form of x. The corrections are applied until another range or line correction statement is found, or an error occurs (see 1.2.6). If an error occurs, the last corrected image is returned and all other change statements are skipped.

The number specified in the x parameter must be greater than that given on any previous range, delete or insert correction statement. The number specified in the y parameter must be equal to or greater than the x parameter.

1.2.5. Change Correction Statements

The change correction statements may be specified in any one of the following four formats.

Format 1: c/new–data

Format 2: c,d/new–data/

Format 3: /old–data/new–data

Format 4: /old–data/new–data/

Format 1 is used to replace the characters of an image from a specified column to the end of the image. The column number is specified in parameter c. Parameter new–data must contain the replacement characters. All of the data following the separator (except trailing blanks) is taken to be replacement characters.

Format 2 is used to replace a specified number of characters in an image. The column numbers entered in the c,d parameters specify the range of characters to be replaced. Parameter new–data contains the replacement characters. If the number of characters in the new–data parameter is greater than the range specified in the c,d parameters, then the characters following the column number specified in the d parameter are right shifted to make room; if it is less, the image is left shifted to close the image.

Format 3 operates similarly to format 1 except that the old–data parameter specifies one or more characters to be replaced. The coding line is scanned and when a find is made, the characters specified by the old–data parameter through the end of the image are replaced by the characters specified in the new–data parameter.

Format 4 operates similarly to format 2 except that the old–data parameter specifies one or more characters that are to be replaced by the characters specified in the new–data parameter.

For Formats 3 and 4, if the line is corrected in ASCII, the character comparison is first done with upper and lower case characters considered unequal. If this test fails, the comparison is done with upper and lower case characters considered equal. When a match is found, the new–data parameter is placed in the line exactly as it was received with no case transformations.

Examples:

```
@ASM,U          PF3.WINDUP,.WINDUP
-30,33-
73/SUBTOT6
42,48/SUBTOT7/
/OVHEAD/CHARG7
/REFUND3/RETURND/
@MAP,IL
```

The U option on the @ASM control statement specifies an update to symbolic element WINDUP. Lines 30 through 33 are to be partially corrected. The characters in columns 73–80 in line 30 are replaced by SUBTOT6. The characters in columns 42–48 in line 31 are replaced by SUBTOT7. The characters OVHEAD in line 32 through the end of the line are replaced by CHARG7. The characters REFUND3 in line 33 are replaced by RETURND. Encountering the @MAP control statement indicates that there are no more corrections to the symbolic element.

1.2.6. Line Correction Diagnostics

When an error occurs, SIR$ passes a print control word in A0 back to the calling processor. The standard action by the processor is to do an ER PRINT$ to inform the user of the error. No other image is returned to the processor on an error return. The formats of the error messages are:

Format 1: SIR EDIT ERR c l

Format 2: c l

where:

c — Indicates the cause of the error. Table 1–1 lists the possible errors.

l — The line number of the last image retrieved from the input element before the error occurred.

Format 1 is returned when a partial line correction error occurred and format 2 is returned when some error occurred within a range or line correction statement.

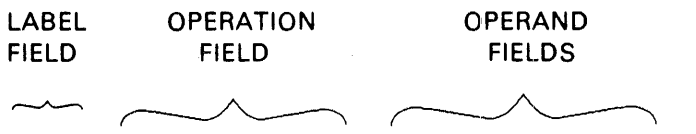
Table 1-1. Partial Coding Line Correction Diagnostics

Error	Description
SEPARATOR	The separator used in the change correction statement is invalid or nonexistent.
COLUMN	The column number specified in a format 1 or 2 change correction statement is out of range; or c > d for a format 2 change correction statement.
NO FIND	The characters given in the old-data parameter of a format 3 or 4 change correction statement could not be found in the line being corrected. NOTE: Whenever one of the above errors occurs, the change correction statement is ignored and the line remains unchanged except Format -x- where the last corrected image is returned.
CARD COUNT <	Not enough change correction statements were provided. Those lines for which no change correction statement was provided remain unchanged.
CARD COUNT >	Too many change correction statements were provided. The excess change correction statements are ignored.
OUT OF SEQ	The range or line correction statement is illegal.
NO CARDS FOLLOW	No cards follow -n where n=l. l is the line number of the last image retrieved from the input element before the error occurred.
INPUT ELEMENT ENDS AT n	To reference a line after the last line of the input element. Line number n being the last line of the element.

1.3. CONTROL STATEMENT SYNTAX

Control statement syntax is described in detail in Volume 2-3.2, for convenience it is described briefly here.

The general EXEC 8 control statement format is as follows:



Brackets are used to indicate optional fields or subfields.

The operation field is terminated by one or more spaces.

The comment field must be preceded by space period space.

The operand fields specify parameters associated with the command fields. These are separated by commas and are specified by the user as dictated by requirements. The content of each operand

field, the number of operand fields, and whether each is required or optional varies with the command selected. Operand fields, in turn, may contain parameter subfields that are separated by various delimiters. For the most part, these subfields are optional within a field. Thus, it is possible to specify parts of a field without specifying the entire field.

When parameter fields and subfields are optional, the following rules apply, where an empty field is defined as one that contains no nonspace characters:

1. Parameter field separators must be specified, left to right, through the last parameter given; fields preceding the last parameter may be empty; trailing field separators need not be specified.

2. The same holds true of parameter subfield specifications within a field.

Leading spaces within a statement are permissible in the following cases:

- Following the master space or at (@) character
- Following a colon (:) when a label is specified
- Following a parameter field (,)
- Following a parameter subfield separator (/)

A space, placed at any position in the coding other than those listed, is interpreted as the termination of the image.

In both batch and demand processing, data images and control statements in a runstream are normally processed sequentially and only upon request by the Executive or by a program operating in that run.

However, a special mode of processing control statements is available during demand processing. This mode directs the Executive to process a control statement immediately after it has been input from a remote terminal. The processing called for by the control statement is also done independently of any current program execution or control statement processing in the runstream. This mode of executing a control statement is specified by a special character, a second @ in column 2 on the control statement. This mode of operation is called transparent mode, and such control statements are called transparent control statements.

Transparent control statements are a subset of the control statement set. The syntax rules for normal control statements, with the following exceptions, also apply to transparent control statements. The exceptions are as follows:

1. The identification of a transparent control statement consists of a @@ versus a @ for a normal control statement.

2. The use of a label on a transparent control statement, while not prohibited, is meaningless.

1.4. FILENAME, ELEMENT NAME NOTATIONS

Filename and element name notations are described in detail in Volume 2-2.6.1 and 2-2.6.4; respectively, for convenience they are described briefly here.

Although the distinction between filenames and element names is often evident from the context, there are many cases where a period must follow a filename or it will be either not accepted, or incorrectly treated as an element name. Therefore, it is best to always specify the period as shown below:

a filename is indicated by: [[qualifier] *]file[(F-cycle)] [/[read-key] [/write-key]].

an eltname is indicated by: [filename.]element[/version] [(element-cycle)]

Qualifier, file, element, and version names are 1-12 alphanumeric Fieldata characters ($ and - characters are also allowed). Keys have 1-6 characters from the entire Fieldata character set, excluding only space, comma, slash, period, and semicolon. F-cycles are numbered upward from 1 to 999; element cycles are numbered upward from 0 to 63.

When the qualifier is omitted, the project-id from the @RUN control statement is used, except in the special case where a leading asterisk appears before the filename and a qualifier has been previously furnished on a @QUAL statement. When the F-cycle or element-cycle number is omitted, the most recently created cycle is used.

When the filename portion of an eltname is omitted, the processor usually assumes an implicit reference to the run's temporary program file, TPF$.

F-cycles may be absolute, indicated by a number, or relative indicated by a -number. A relative F-cycle of (+1) must be used to distinguish a newly assigned 'to be catalogued' file (see @ASG,C and U options) from an existing catalogued file of the same name. A relative F-cycle of (-3) would designate the fourth oldest file that was catalogued under the specified filename. Element cycles are referenced by their actual number, such as (0) or (6), or by relative number such as (-2).

1.5. SOURCE INPUT/OUTPUT ROUTINE CONTROL OPTIONS

Source input/output routine (SIR$) options are described in detail in Volume 4-2.1.4. For convenience, the table of control options is reproduced here.

Table 1-2 contains a list of those options used by the source input/output routine to control the input and output of the souce language elements. Most Sperry Univac supplied language processors (FTN, MASM, ACOB, NUALG, and so forth) use the source input/output routine to obtain their input. Therefore, the listed options are generally applicable to these language processors.

Table 1-2. Source Input Routine Options

Option Character	Description
G	Input is compressed symbolic in columns 1-80 of the card deck.
H	Input contains sequence numbers in columns 73-80 of the symbolic images.
I	Insert a new symbolic element into the program file. I is not permitted when applying corrections to an element.
J	Input contains compressed symbolic images in columns 1-72 of the cards and sequence numbers in columns 73-80. These sequence numbers are not checked by the K option.

Table 1-2. Source Input Routine Options (continued)

Option Character	Description
K	Check sequence numbers in columns 73-80 of the symbolic images (valid only with H option).
P	Output symbolic element in Fieldata. (Compare with Q.)
Q	Output symbolic element in ASCII. (If neither P or Q is specified, code type of input element, if any, is used; otherwise, the code type is based on the type of call to SIR$, ASCII if GETAS$ and Fieldata if GETSR$.) If both P and Q are specified, output symbolic element with mixed images can be in Fieldata and ASCII.
U	Update and produce a new cycle of the symbolic element.
W	List correction lines.

1.6. PROCESSOR CONTROL STATEMENTS

Two separate system library files are available during the processing of a user run; the absolute library file (SYS$*LIB$) and the relocatable library file (SYS$*RLIB$).

The absolute library file (LIB$) contains the absolute elements of each standard processor included with the operating system. LIB$ may contain any other processor and executable program added by the installation.

The relocatable library file (RLIB$) contains the system-supplied relocatable elements and procedure elements which may be needed to assemble, compile, or collect the user program.

A temporary program file (TPF$) is created by the Executive for each run that is initiated. The qualifier for the filename is taken from the project-id field of the @RUN control statement. The file may be used as a scratch file for the user program's symbolic, relocatable, and absolute elements. Note, however, that since this is a temporary file it is discarded at run termination. Demand mode users will find that it is safer to keep 'work in progress' in a catalogued file so that this work will not be lost in case of unplanned run termination.

The general format of the processor control statement is:

```
@label:processor,options    param-1,param-2,param-3,...,param-n
```

The label field is as described in Volume 2-3.2.1. The processor field is the name and file location (see Volume 2-3.2.2) of the absolute element desired. The following is an example of a generalized processor control statement where the processor is located in a user-specified file rather than in the system library file LIB$:

```
@USER*FILE.PROG/ABS,P                FILE.IN, ELTOUT, FILE.OUT
```

All of the standard processors must be called using the above call form (rather than @XQT) since they expect to retrieve the parameters on the call line. Such parameters cannot be retrieved on an @XQT call (see 2.3).

The rules for locating the element in the processor field are slightly different from the standard rules for locating an element specified on an @XQT control statement. The processor call rules are:

1. If a filename is specified, then that file is searched for the absolute element.

2. If a filename is not specified but there is a leading period, then TPF$ is searched for the element; if there is no find, LIB$ is searched.

3. If a filename is not specified and there is no leading period, then the system library file SYS$*LIB$ is searched for the element; if there is no find, then TPF$ is searched. The abbreviations for the standard processors (ACOB for COBOL, FTN for FORTRAN, and so forth) are the names of the respective absolute elements.

In general the option field has meaning only for the particular processor, though there are some options that have the same meaning for all processors. The format of the options parameter is described in Volume 2-3.2.2.

The param-1, param-2,...param-n parameters contain information supplied to the processor. With the exception of the DATA processor (see Section 6), which works only with SDF files and, therefore assumes filenames, the parameter fields are assumed to be in element name form although they need not represent element names. The meaning of the parameter fields is determined by the processor. The following rules are followed by the processors supplied by Sperry Univac:

1. If a field intended to contain the name of a program file is not specified, TPF$ is assumed.

2. If a field is to contain an element name and the element name is specified but not the filename and there is no leading period, TPF$ is assumed. If there is a leading period, then the filename is taken from previous field provided that the field exists and was intended to name an element or a program file.

The source language processors (MASM, ACOB, FTN, NUALG, and so forth) have a common interpretation of several options as well as the first three parameters. The typical standard language processor control statement takes the form:

```
@ACOB          SI,RO,SO
```

where SI, RO, and SO represent eltname-1, eltname-2, and eltname-3.

The meanings of these parameters are:

SI (Source Input)

If no new element is being introduced this parameter specifies the source of input for the processor.

If a new symbolic element is being introduced from the runstream (I-option set), this parameter specifies the file into which the new element is placed and the name which it is given. If an update is being performed (U-option set), then this parameter specifies the element and the cycle of the element being updated.

It is possible to specify a symbolic element from a tape file for this parameter. The tape file must be in element file format (see 11.2.2) and the file must be positioned (@FIND) so that the element label is read in. Corrections to a symbolic element from a tape are permitted provided that the output is a symbolic element in a program file.

RO (Relocatable Output) — This parameter specifies the name and the program file into which the element produced by the processor is placed. There is no restriction on the type of element being produced. For example, most of the processors produce relocatable binary elements; the collector produces either absolute or relocatable binary elements.

SO (Source Output) — This parameter specifies the name and the file for the updated symbolic element if no U-option.

The System Relocatable Library routine PREPRO examines the facility description of each file specified in the SI, RO, and SO fields and assigns those files which do not meet the minimum assignment requirements.

SI - assigned

RO - exclusively assigned

SO - exclusively assigned

The RO and SO fields are assigned exclusively because the language processor will modify the table of contents of the file(s) and write in the text portion of the file(s). If another run has any of the files assigned in such a way as to prevent PREPRO from obtaining the minimum assignment the processing will be aborted if the run is in demand mode or held until the file is available if the run is in batch mode. Availability of required files may be checked by assigning the files, with the minimum assignment required, before calling the processor.

The source language processors do not interpret the SI, RO and SO fields to determine uniqueness or duplication of names. If an element name in the SI, RO or SO field matches an already existing element in name/version and type, and the field is an output field the old element will be replaced with the new element.

If no element name is specified for RO and SO or the parameter is left blank the following rules apply:

1. If there is no file information and the parameter does not have a preceding period or if the parameter is void, then the file specified in the SI parameter is assumed.

2. The element name from the SI parameter is assumed.

3. If there is no version specified, then the version from the SI parameter is used.

Tables 1-3 and 1-4 describe the valid possibilities. The I and U options along with the SI parameters determine the interpretation of the processor control statement. An error message is printed if there is any deviation from these rules. Table 1-3 is valid for the MASM, ACOB, FTN, NUALG, MAP, and CFOR language processors (require SI, RO, and SO); Table 1-4 is valid for the PDP and ELT processors (require only SI and SO).

Table 1-3. Processors That Use the SI, SO, and RO Parameters

I or U Option	SI Notes	RO Notes	SO Notes	Element Produced
Neither or I Only	Not specified	This parameter may or may not be specified. If it is not specified, NAME* is assumed. It is invalid to specify a cycle.	Illegal to use this parameter.	New relocatable element.
Neither	Parameter is completely specified.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.	If void, no source output is produced. If this parameter is not completely specified then information from the SI parameter is used.	New relocatable element. Also new symbolic element if SO was specified.
I Only	Parameter is completely specified but without a cycle.	If not completely specified, the information from SI parameter is used. It is invalid to specify a cycle.	Illegal to use this parameter.	Relocatable and symbolic elements.
U Only	Parameter must be completely specified.	If this parameter is not completely specified, then the information from the SI parameter is used. It is invalid to specify a cycle.	Not required, information from the SI parameter is used. When SO is not specified, the SI element is updated to produce the next higher element cycle from the SI cycle specified, or the latest cycle if no cycle was specified. If SO is specified, the next higher element cycle is created and transferred along with all previous cycles up to the cycle maximum to the new SO element; SI element remaining unchanged. In either case, it is invalid to specify a cycle for SO.	New relocatable element and updated symbolic element if SO was specified.

Table 1-4. Processors That Use the SI and SO Parameters

I or U Option	SI Notes	SO Notes	Element Type Produced
Neither	Not specified.	Illegal to use this parameter.	The runstream input is processed and listed but no element is produced.
Neither	Parameter is completely specified. If SO is void the L option is assumed.	If this parameter is void, no source output is produced. If it is not completely specified, then the information from the SI parameter is used. Invalid to specify a cycle.	Update (no cycling).
I Only	Not specified.	Illegal to use this parameter.	No element is produced.
I Only	Parameter is completely specified but without the cycle.	Illegal to use this parameter.	New element.
U Only	This parameter must be completely specified.	If this parameter is not completely specified, then the information from the SI parameter is used. When SO is not specified, the SI element is updated to produce the next higher element cycle from the SI cycle specified, or the latest cycle if no cycle was specified. If SO is specified, the next higher element cycle is created and transferred along with all previous cycles up to the cycle maximum to the new SO element; SI element remaining unchanged. In either case, it is invalid to specify a cycle for SO.	Update (with cycling) if SO was specified.

2. Program Construction and Execution

2.1. INTRODUCTION

The SPERRY UNIVAC 1100 Series Executive System provides the ability to combine the relocatable elements generated by a language processor into an executable (absolute) element. This combination or collection of relocatable elements is done by a system processor, the Collector. The absolute element produced by the Collector is structured such that the Executive program loader can place it in execution. Once the absolute program has been created (that is, collected), it may be saved and reexecuted many times. The program need only be recollected when a modification to it is desired.

An absolute element (program) is placed in execution through use of a program execution control statement (@XQT or processor call) within the control stream. When an @XQT or processor call control statement is encountered by the Executive, the program is retrieved from its mass storage file, loaded into main storage, and execution is initialed. If the statement was a processor call the EXEC presents (in table form) the parameter field from the call when the first symbiont input (READ$) request is made. If @XQT is used the first READ$ obtains the first data image following the @XQT.

During execution, a program can control which parts of the absolute elements are in main storage by requesting the Executive to load previously-defined program overlay segments or by linking to program banks. In addition, the program has the capability of attaching to or linking to other previously defined absolute elements. This program structure supports the dynamic sharing of both code (usually reentrant) program banks and banks containing data between multiple users. Such shared banks are termed common banks.

2.2. THE COLLECTOR

The Collector is called by the @MAP processor control statement (see 2.2.1). It provides a direct means of collecting and interconnecting relocatable elements to produce a program in an executable form. This form is called an absolute element. Optionally, the Collector can be used to create a single relocatable element from a collection of several relocatable elements. The three basic inputs to the Collector are:

- The parameters supplied on the @MAP control statement
- The information supplied by the collector directives

- Relocatable elements taken from various sources, such as:
  - the Temporary Program File (TPF$)
  - user-created program files
  - the System Relocatable Library (SYS$*RLIB$)

The three basic outputs of the Collector are:

- An absolute or relocatable element
- A symbolic element
- A program listing

The primary output of the Collector is the relocatable or absolute element which results from the collecting and linking of the various relocatable elements. This element is given a name and placed within a program file for subsequent use. Both the element name and the file in which the element is placed may be dictated by the user.

Usually the Collector includes within an absolute element a set of tables for use by the diagnostic system. This output can be suppressed by the user (see Table 2-1).

The Collector normally divides the absolute element into two logical structures called banks. The first bank, called the I-bank, is comprised of information which is contained under the odd location counters of the relocatable elements to be included in the collection; the second bank, called the D-bank, contains information contained under the even location counters of the included relocatable elements. By convention, but not necessity, the instructions of a program are placed under odd location counters, while the data portions are placed under even counters. This allows a separation of instructions and data to be achieved, with instructions directed to the I-bank and data to the D-bank.

There are times when the user may wish to structure a program into logical entities or banks which are different from the single I-bank, D-bank normally produced. To do this, the user must explicitly name the bank or banks comprising the program and then direct the Collector as to the program portions to be contained in each bank. This method of collection, in which a program's banks are explicitly named, will generally be called a 'bank-named collection'.

When no banks are explicitly named, the Collector will generate one I-bank of odd location counters and one D-bank of even location counters, and the collection will be referred to as a 'bank-implied collection'.

2.2.1. Collector Initiation (@MAP)

Purpose:

Specifies that the Collector is to combine a set of relocatable elements into one absolute or relocatable element. All parameters in the @MAP control statement are optional. See Volume 2-3.9 for additional information regarding processor control statements.

Format:

@label:MAP,options eltname-1,eltname-2,eltname-3

Parameters:

options — See Table 2-1 and source input routines options (see Table 1-2).

eltname-1 — Specifies the input symbolic element which contains the Collector source language (see Volume 2-3.9).

eltname-2 — Specifies the absolute output element. When the R option is specified, eltname-2 names the relocatable output element (see Volume 2-3.9).

eltname-3 — Specifies the output source language element (see Volume 2-3.9).

Table 2-1. @MAP Control Statement, Options

Option Character*	Description
A	Under no circumstances is the error exit (ERR$) to be taken during the collection, even if the collection is destroyed.
B	Mark the absolute element so that the program area is not cleared to zero prior to loading the program and any indirectly loaded segments (see 2.2.4.5.2).
D	Print a diagnostic message for all possible addresses over 65K (0177777). Check for certain possible instruction format violations.
E	Allow program addresses to exceed 65K (0177777). If this option is omitted and the program's D-bank exceeds 65K, the D-bank starting address is moved downward so that all (or as many as possible) of the over-65K addresses are forced below 65K. In bank-named collections, all possible D-bank starting addresses are moved downward.
F	Mark the output absolute or relocatable element as quarter-word sensitive. (Also see T option.)
L	Produce a complete listing which contains the following information concerning the program area: - main storage allocated to each element and segment - program address of all external definitions - the symbol '?' following any undefined entry point - the scale drawing of program segmentation and bank structure - the external references of each element
N	Produce the most abbreviated print listing available.
R	Generate a relocatable element instead of an absolute element.

Table 2-1. @MAP Control Statement, Options (continued)

Option Character*	Description
S	Produce a summary listing which includes a scale drawing of program segmentation and bank structure.
T	Do not mark the output element as quarter-word sensitive. If neither the T nor F options are specified, the program sensitivity is determined as follows: - if only third word sensitive elements and elements with neither T nor F sensitivity are present, T is used. - if only quarter-word sensitive elements and elements with neither T nor F sensitivity are present, F is used. - if both third- and quarter-word sensitive elements are present, the sensitivity of the element containing the program starting address is used. - if both third- and quarter-word sensitive elements are present, the sensitivity of the element containing the program starting address as specified on the ENT directive is used. If the ENT directive is not used, the absolute element is marked insensitive.
V	Assign all addresses but strip off all D-bank code (can be used to create a reentrant processor - see 2.2.3.5). Compare with Y option.
X	If an error is detected, terminate the collection and exit via ERR$. When the X option is omitted, the results of the collection are accepted, even though there may be minor errors, as long as an absolute element is produced. This option is assumed when collector is automatically called by @XQT.
Y	Assign all addresses but strip off all I-bank code. Compare with V option.
Z	Suppress generation of diagnostic tables in the absolute element which are used by diagnostic system.

* Also see source input routine (SIR$) options, Table 1-2.

Examples:

```
1.  @MAP
2.  @MAP            OLDFILE.OLDELEMENT,A,NEWFILE.NEW/ELEMENT
3.  @MAP            SYMIN/C,BACKUP.ABSOUT
4.  @MAP,I          BACKUP.SYMOUT,.ABSOUT
5.  @MAP,U          SYMIN(3),ABSOUT/REVISED
6.  @MAP,IRXLD      ARB,ARB
```

1. This @MAP control statement produces the same results as the @MAP,I control statement. The names for the symbolic and absolute elements are automatically assigned by the Collector (see Volume 2-3.9). The printed output and internal table entries would appear as if the control statement had been: @MAP,I TPF$.NAME$. If no directives follow, the directive IN TPF$. is assumed.

2. Element OLDELEMENT from file OLDFILE is updated by any source language statements following the @MAP control statement. The output source language goes into file NEWFILE, element NEW/ELEMENT. The absolute element A goes into TPF$.

3. Version C (latest cycle) of element SYMIN from TPF$ is the input symbolic element. The absolute output element ABSOUT is written in file BACKUP. No source language output is produced; any successive correction lines are applied but not saved.

4. The source language statements (Collector directives) following the @MAP control statement are inserted as the symbolic output element SYMOUT in file BACKUP. The absolute element ABSOUT is also put into file BACKUP.

5. Cycle 3 of element SYMIN is updated to produce cycle 4 of element SYMIN which is located in TPF$. Any correction lines are saved. Version REVISED of absolute output element ABSOUT is also put in TPF$.

6. The source language statements following the control statement become the symbolic element ARB in the TPF$ file. The output element goes into TPF$ as relocatable element ARB. If errors are encountered during the collection, the run is terminated. A full listing is produced. Diagnostics are printed for addresses over 65K.

2.2.2. Collector Directives

The Collector directives enable the programmer to control the collection of his program. These directives:

- are free-form; hence, they may begin in any column of the source language image. For rules regarding the presence of blanks, see Volume 2-3.2.6.
- may contain comments preceded by the blank-period-blank construction.
- follow the standard dropout rules (see 1.4 and Volume 2-2.6.6) pertaining to filenames, element names, and so forth.

For the collection of complex programs which require relocatable input from many sources, construction of overlay segments, the use of multiple libraries, or the construction of multiple banks, the user must prepare a set of Collector directives. These statements may follow the @MAP control statement or be contained in an element in a program file which is specified as input on the @MAP statement. The user has the same access and updating facilities for the (@MAP) symbolic element as for any other type of symbolic element.

Certain Collector directives can be used only in bank-named collections. These are IBANK, DBANK, FORM, and $lcs (location counter set). The second format specified for the IN and the second, third and fourth formats of the LIB directives can be used only in bank-named collections.

On all Collector directives, in all fields and subfields, a parameter consisting of a set of characters contained within quotes, e.g., 'XYZ-CAT', will be treated as an alphanumeric name. Any normal termination characters will simply be considered characters within the name.

2.2.2.1. Element Inclusion (IN)

Purpose:

Allows the user to specifically include an element or all relocatable elements of a file in the collection of a program. An element may be preceded by a filename. The elements indicated on an IN directive are placed in the segment named by the preceding SEG directive (see 2.2.2.14).

All parameters in the IN directive are optional. If all parameters are omitted, IN TPF$. is assumed.

Format:

```
IN    name-1,name-2,name-3,...,name-n
IN(bank-list)    name-1($lcs),name-2($lcs),...,name-n($lcs)
```

Parameters:

name	Specifies the element or entire file to be included in the collection. See 1.3, 1.4 for standard file and element notation.
bank-list	A list of bank-names to be used with local element inclusion.
$lcs	Specifies which location counters of name-1 are to be included in this part of the program.

Description:

Bank-list and $lcs are used only with bank-named collections (see 2.2.5).

By stating FILENAME without a following element name, the user can specify the inclusion of all relocatable elements in a program file. In bank-implied collections if some elements of a file have been explicitly named, these are included as specified and the 'IN' filename serves to bring in only the remaining elements in the file. When specifying an entire file for inclusion, a period must follow the filename.

When name consists of an element name only with no version specified, any RB element with that name irrespective of version, is eligible for inclusion. If RB elements with the same element name but different version names exist in the same file, ambiguities may arise. The CLASS directive (see 2.2.2.8) may be used to overcome the ambiguity. Alternatively, the ambiguity will not be present if the version name is explicitly stated as part of name on the IN statement.

```
IN    name/
```

must be used to specifically select an RB element with a version name consisting of 12-space characters, which is the default version.

Elements named, but not directly associated with a filename, are searched for first in TPF$, then in any files named on LIB directives (see 2.2.2.3) and finally in the System Relocatable Library (RLIB$).

In bank-implied collections, an element name may appear on only one IN directive and only once during a collection. A version may be present, but the element name itself may still appear only once. The version is necessary only if needed to distinguish between elements with the same basic name but different versions. See 2.2.5.6.2 for including an element more than once in a bank-named collection.

Common blocks may be named on IN directives, but must not have an associated implicit or explicit filename because they are imbedded within other elements. For inclusion of a common block in the collection, see 2.2.4.6.

For the order of elements explicitly and implicitly included in the collection, see 2.2.3.2 and 2.2.5.7.

For either format FRSTIN may be used instead of IN, as the first element inclusion statement in a bank-implied collection or after a BANK or SEG statement. When used it specifies that the first named element is to be positioned at the beginning of the segment it is collected in.

Examples:

```
1.  IN  FILEA.,FILEB.
2.  SEG             ADAY1  .
    IN  FILEB.BB,.CC,DD
3.  SEG             BDAY2  .
    IN  COLLECTOR*F8(1).INIT/REV
```

1. All relocatable elements in FILEA and FILEB are included in the collection.

2. Elements BB and CC from file FILEB and element DD, whose filename is not indicated, are included in the collection of segment ADAY1.

3. The relocatable element INIT version REV in file COLLECTOR*F8 F-cycle 1 is included in the collection of segment BDAY2.

2.2.2.2. Element Exclusion (NOT)

Purpose:

Names the elements which are to be excluded from the entire collection.

All parameters in the NOT directive are optional. If all parameters are omitted, NOT TPF$. is assumed.

Format:

```
NOT name-1,name-2,...,name-n
```

Parameters:

name	Specifies the element, with or without filename, or file to be excluded from the collection. If the version name or filename is omitted, all elements of the specified name are bypassed.

Description:

When all elements of a file are to be excluded, the entire file may be designated for exclusion with a NOT directive. The effect of NOTing a whole file is to make the file inaccessible for searching as a library file. A period must follow the filename to ensure that it is not interpreted as an element name.

If a filename with no following element names appear in a NOT directive, elements within the named file can be explicitly included during the collection. This is useful only with TPF$ and SYS$*RLIB$, since these are the only two files which are normally automatically searched.

Examples:

```
1.  @MAP,I          A,A
    NOT             CWW,LRR
2.  MAP,I           A,A
    IN  FILEA.
    NOT             FILEA.CL,.AB
3.  @MAP,I          A,A
    IN  FL1.,FL2.
    NOT FL1.XXX,.XX2,FL2.CAT,.CAT2
4.  @MAP,I          A,A
    IN  FIL1.
    NOT SYS$*RLIB$.
    NOT TPF$.
```

1. All elements named CWW and LRR are excluded from the collection; all other relocatable elements in TPF$ are included.

2. All elements named CL and AB from FILEA, are excluded from the collection; all other relocatable elements in the file FILEA are included.

3. Elements XXX and XX2 from file FL1 and elements CAT and CAT2 from file FL2 are excluded from the collection; all other relocatable elements from files FL1 and FL2 are included.

4. Relocatable elements from SYS$*RLIB$ and TPF$ are excluded from the collection. All elements from FIL1 are included.

2.2.2.3. File Search Sequencing (LIB)

Purpose:

Specifies which files (libraries) are to be searched by the Collector prior to searching the System Relocatable Library SYS$*RLIB$. Each file named on a LIB statement must have an entry point table created by the @PREP FURPUR function (see 4.2.11), if the file contains elements which will be implicitly included to satisfy external references.

All parameters in the LIB directive are optional, except filename-1.

Format:

```
LIB     filename-1,filename-2,...filename-n
LIB     filename-1(bankname/$lcs,bankname/$lcs,...).filename-2...
LIB     (bankname/$lcs,bankname/$lcs...)
LIB     filename-1( ),filename-2( ),...filename-n( )
```

Parameters:

filename	Specifies the files to be searched, named in the order in which they are to be searched. Those files containing elements not explicitly included in the collection but which define symbols required to satisfy external references, must have been prepped (@PREP).
bankname	Specifies a bank in which implicitly included elements are to be placed.

$lcs	Specifies which location counters of the implicit elements go to corresponding banks.
( )	The parameters specified on the last preceding LIB (bankname/$lcs,...) directive are to be applied for the filename specified.

Description:

The second, third and fourth formats above and parameters bankname and $lcs are used only in bank-named collections (see 2.2.5.7).

The specified files are searched for elements named for inclusion from that file or from an unspecified file, and when all explicitly included elements have been found, or searched for and not found, the files are searched in the order in which they appear in the directive for satisfying external references. A file may be searched more than once if the filename appears more than once on a LIB directive. A file is searched only once for each time it is specified on a LIB directive.

When several LIB directives are given they have a cumulative effect. For example, assume file A has external references satisfied by elements in file B which in turn have external references satisfied by elements in file A. If the elements are not explicitly included by IN directives, the following directive is necessary to ensure the inclusion of all referenced elements:

```
LIB    A,B,A
```

RLIB may be used instead of LIB in all formats. However, the effect in format 3 is lost. Any element from a file on such a statement is marked as being from SYS$*RLIB$. The result is that these elements will not be dumped by @PMD unless the 'L' option is used with the PMD request, nor will they be traced by SNOOPY.

Examples:

```
1.  LIB        CHR1
2.  LIB        USE1,USE2,USE3,USE1
```

1. File CHR1 is searched after TPF$, and before files USE1, USE2, USE3, USE1 (a second time) and the System Relocatable Library.

2. Files USE1, USE2, USE3, and USE1 (a second time) are searched in that order after TPF$ and CHR1 and before the System Relocatable Library is searched.

2.2.2.4. External Definition Retention (DEF)

Purpose:

Specifies entries in the ENTRY$ table. This table contains all the locations and names of the external definitions retained after the collection of the absolute or relocatable element.

NOTE:

The DEF and REF (see 2.2.2.5) directives are primarily useful in the collection of reentrant processors and with R option collections.

All parameters in the DEF directive are optional.

Format:

```
DEF    def-1,def-2,def-3,...,def-n
```

Parameters:

defs — Specifies the external definitions to be retained.

Description:

The DEF directive causes the Collector to build an external definition table and a COMMN$ table which defines the common blocks in a program. The user can address the two tables by the Collector-defined names ENTRY$ and COMMN$, respectively (see 2.2.8). The COMMN$ table will also be built if the DEF directive is present with no parameters.

If the R option was given on the @MAP control statement, the DEF directive must be used to specify those externalized labels which are to remain externalized in the merged relocatable output.

If no element explicitly named in an IN directive contains the named external definition, a search of the library files (see 2.2.2.3) is made to find an element in which the symbol is defined.

Example:

```
DEF          SIN,COS,SORT
```

The listed external definitions, SIN, COS, and SORT and their locations are retained after the collection in the ENTRY$ table of the resultant element.

2.2.2.5. External Reference Retention (REF)

Purpose:

Creates a list of external references to be retained by the resulting absolute or relocatable element. No attempt is made to satisfy any references made to names indicated on REF directives. The table of retained external references is program addressable by the Collector-defined name XREF$.

All parameters in the REF directive are optional.

Format:

```
REF    ref-1,ref-2,ref-3,...,ref-n
```

Parameters:

refs — Specifies the external references to be retained.

Description:

If an external definition that is identical to a REF name is encountered, a diagnostic is printed and the external definition is ignored.

The REF directive causes the Collector to build the COMMN$ table in the same manner as the DEF directive (see 2.2.2.4).

Example:

```
REF          VALUE1,VALUE2,SUBROUTINE4
```

The listed external references (VALUE1, VALUE2, and SUBROUTINE4) are retained after the collection in the XREF$ table of the resultant element. Any references to these symbols are satisfied with the address of word 3 of the appropriate entry in the XREF$ table (see 2.2.8).

2.2.2.6. Starting Address Redefinition (ENT)

Purpose:

Provides the user with the capability of overriding any start addresses provided by relocatable elements. Upon program initialization, control is transferred to the absolute address of the named symbol.

Format:

```
ENT name
```

Parameter:

name	Must be an externally defined symbol which is the newly defined starting point.

Description:

In the absence of an ENT directive, the first start address encountered in processing the relocatable elements becomes the program start address. The start address must be contained in the main segment as this is the only segment initially loaded at execution time. In a bank-named collection, the start address must be contained in the main segment of an initially-based bank (see 2.2.5.2).

Example:

```
ENT          GGYP
```

Control is passed to the absolute address of the symbol GGYP in the main segment.

2.2.2.7. External Reference Definition (EQU)

Purpose:

Provides the means to assign, during the collection, a value to an undefined symbol or to change the value of an external definition.

All parameters in the EQU directive are optional except name-1/value-1.

Format:

```
EQU    name-1/value-1,name-2/value-2,...,name-n/value-n
```

Parameters:

names — Specifies the symbols to be defined

values — The values to be assigned to the preceding name parameters.

The assigned values may be as follows:

- octal integers (indicated by a leading zero)
- decimal integers
- a symbol
- a symbol with an offset (for example, BOB+4)

Description:

Any symbol used in the value parameter must be externally defined in one of the elements specified for inclusion in the collection. If an external definition which duplicates an EQU name is found, the external definition is ignored and a diagnostic is printed. In bank-named collections, only global symbols may be used on the EQU directive (see 2.2.5.6.2).

Examples:

```
1.  EQU     JOE/0200
2.  EQU     AL/BOB+4
3.  EQU     JOE/0200, ABE/SAM+10
```

1. The external reference JOE is defined as 0200.

2. The external reference AL is defined as the value $BOB+4_{10}$.

3. The external references JOE and ABE are defined as 0200 and $SAM+10_{10}$, respectively.

2.2.2.8. Element Selection Determination (CLASS)

Purpose:

Uniquely specifies one element version in a program file when more than one element has the same basic name but different version names. In the collection this occurs when:

- The version of the element was not specified on an IN directive and more than one relocatable element has that name.
- More than one relocatable element defines an external reference.
- A filename was not specified with the element on an IN directive and the element with different version names is present in more than one file.

Format:

```
CLASS     string
```

Parameter:

string — Consists of 12 alphanumeric characters, asterisks, $, -, and blanks representing the versions of the elements. The string begins with the

first nonblank character following CLASS, and is terminated after 12 characters, or by the space-period-space comment delimiter. If fewer than 12 characters have been processed upon termination of the string, significant blanks are filled in on the right to make a 12 character string.

Description:

Successive CLASS directives have a cumulative effect and different ordering of CLASS directives may give different results.

Asterisks in a string represent character positions in the version name to be ignored. Blanks in a string are valid.

When several elements qualify to be included in the collection, the Collector compares the string parameter in the CLASS directive with the version names of the available elements. If the element version name is not identical to the string parameter, it is not included in the collection.

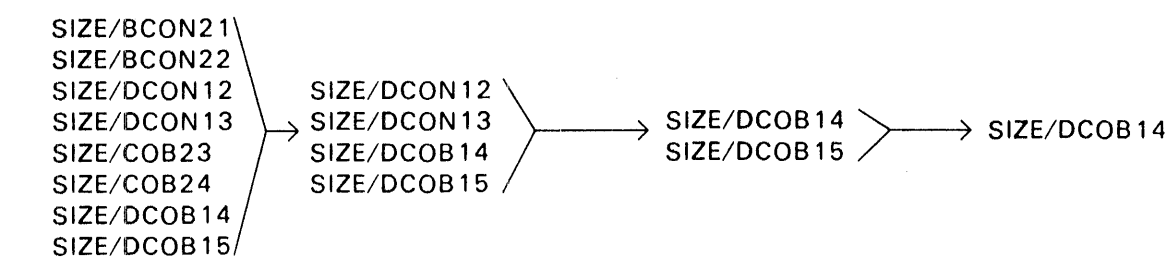
If, after the first comparison, more than one element qualifies, the string in the next CLASS directive is used in eliminating the remaining versions.

If all the CLASS directives have been used and there still remain more than one qualifying element, none of the remaining elements is used in the collection; a diagnostic message is given.

Example:

```
1.  @MAP,I          SAMP,ALPHA
    SEG AARD
    IN  SIZE
    CLASS           D**********
    CLASS           ***B********
    CLASS           *****4******
2.  @MAP,I          SA
    IN  ELT1
    CLASS           D*LA********
```

1. The IN directive does not specify which version of the element SIZE is to be used in the collection. The three CLASS directives specify that the version DCOB14 be used in the collection. Graphically this can be shown as follows:



2. The CLASS D*LA******** directive specifies that version D2LARGE of ELT1 element be used in the collection. Graphically this can be shown as follows:

The IN ELT1 directive selects the following elements:

The CLASS D*LA******** directive makes the final element selection:

2.2.2.9. Corrections for a Relocatable Element (COR)

Purpose:

Specifies that a correction to a relocatable element is to be incorporated into the absolute element produced with the original relocatable remaining unchanged. A COR directive may replace any text word with one of the following:

- an instruction word
- a data word
- a data word containing up to two symbols or values representing the upper and lower halves of the word

Format:

```
COR   eltname
```

The correction data images which follow the COR directive may be in any one of the following formats:

```
address,lc-1   f j a x h i u,lc-2,eltname-1
address,lc-1   dataword
address,lc-1   data,lc-2 data,lc-3
```

Parameter for the COR directive:

eltname	Specifies the element to which the corrections are to be made.

Parameters for the correction data images:

address,lc-1	Specifies the relative address and location counter of the relocatable element to which the corrections are being made.
f j a x h i u	Specifies field values in instruction word format that are to be inserted. Blanks are used to separate each part of the instruction correction. The u-field may be a: - symbol - symbol and offset - octal number (a zero must precede the number) - decimal number (no preceding zero)

dataword	Specifies a numeric value.
data	Specifies a symbol, symbol and offset, octal, or decimal correction.
lc-2 lc-3	Specifies that the u- or data fields are relative to the value of the specified location counter. If omitted, the fields contain absolute values.
eltname-1	Specifies element to which the location counter (lc-2,lc-3) belongs; if omitted, the element being corrected is assumed.

Data fields, including the u-field in format 1, may be specified as negative by the presence of a leading minus (-) sign.

Description:

The corrections contained in a COR directive are ignored if the R option was specified on the @MAP control statement.

Any number of correction statements may follow the COR directive.

COR statements cannot contain instructions which are jumps to any indirectly-loaded segment.

A symbol must be an externalized entry point.

In bank-named collections, a COR directive and its parameters may apply only to global elements (see 2.2.5.6.2).

Examples:

```
1.  COR             ELT1
2.  000001,01       051 00 00 00 0 0 000011,01,ELT2
3.  000004,01       006 00 015 00 0 0 DUMMY+1
4.  000007,01       0000000000114
5.  000011,01       00000111 DUMMY+1
6.  000006,02       0000112,01    0000013
7.  000011,02       DUMMY+2
8.  000014,02       02,02    DUMMY+2
9.  000016,03       027 00 017 00 0 0  01176,02
```

1. Corrections to the relocatable element ELT1 are to be applied to the final absolute element.

2. The word being collected under element ELT1 at relative address 01, location counter 1, is modified to set function code to 051, and the j-, a-, x-, h-, and i-fields to zero. The u- field is given the value of 011 relative to location counter 1 of element ELT2.

3. The word appearing at relative address 04 under location counter 1 of ELT1 is modified to set function code to 06; a field to 015 and u- field to DUMMY+1. The j-, x-, h-, and i- fields are set to zero.

4. The word appearing at relative address 07 under location counter 1 is being changed to contain 0114.

5. The word at relative address 011 under location counter 1 is changed to contain 0111 in H1 and DUMMY+1 in H2.

6. The data at relative address 06 under location counter 2 contains 0112, relative to location counter 1 of ELT1, in H1, and 013 in H2.

7. The data at relative address 011 under location counter 2 receives the value of symbol and offset, DUMMY+2.

8. The data at relative address 014 under location counter 2 receives the data 02 relative to location counter 2 in H1, and the value of symbol DUMMY+2 in H2.

9. The instruction word appearing at relative address 016 under location counter 3 receives the function code 027; the a-field 017 and u- field of 01176, relative to location counter 2 of ELT1. The j-, x-, h-, and i- fields are set to zero.

2.2.2.10. Adding Snapshot Dumps (SNAP)

Purpose:

Specifies the elements in which a snapshot dump is to be taken. The snap data images immediately following the SNAP directive specify the address within the element where the SNAP is to be taken, length of the dump, and the frequency of the dump.

SNAP Directive Format:

```
SNAP  eltname
```

Parameter:

eltname	Specifies the element name where the dump is taken.

SNAP Data Image Format:

```
address-1,lc-1   address-2   length,registers   times,frequency
```

All parameters are optional except address-1, lc-1, address-2, and length.

Parameters:

address-1,lc-1	Specifies the address and the location counter within the relocatable element of the instruction at which the dump is to be executed. This field may not contain a symbol.
address-2	Specifies the starting address of the program area to be dumped. This field may be in the form address, location-counter, element-name or symbol + offset.
length	Specifies the length in words of the program area to be dumped.
registers	Specifies which registers are to be dumped. The following codes are used: 00 - No registers dumped 01 - Only R registers dumped

02 – Only A registers dumped

03 – Both A and R registers dumped

04 – Only X registers dumped

05 – Both X and R registers dumped

06 – Both X and A registers dumped

07 – A, X, and R registers dumped

times — Specifies the maximum number of times the snapshot dump is to be taken. If omitted, the value of 100_{10} is assumed.

frequency — Specifies at what frequency of reference the dump is to be taken. If omitted, the value 1 (which dumps every time the SNAP is encountered) is assumed.

Description:

No more than 16 snapshot dumps may be requested in any one collection. If more than one snapshot of the same element is to be taken, multiple data images may follow the SNAP directive.

When the dump request instruction SLJ SNAP$ is inserted at a specified address, the instruction appearing there is placed in a table in element SNAP$. After the dump is taken, the saved instruction is executed from within SNAP$ as if it had not been moved. If the saved instruction is a jump instruction, control transfers immediately to the location specified in the jump instruction; otherwise, control is transferred to the location following the location from which SNAP$ was called. Because the replaced instruction is executed from within SNAP$, the replaced instruction:

- Must not be altered during program execution.
- Must not be referenced as data or by indirect addressing.
- Must not be an SLJ instruction which specifies indirect addressing or indexing.
- Must not be an LMJ instruction which specifies indirect addressing or indexing.
- Must not be an LIJ or an LDJ instruction.
- Must not be an EX instruction which references an LMJ or SLJ instruction.
- Must not be a test and skip instruction.
- Must not be used in reentrant code.

In a bank-named collection, the SNAP directive and its parameters may apply only to global elements and entry points (see 2.2.5.6.2).

Example 1:

```
1.  SNAP          LARK
2.  010,1         012,1   020,07     0200,010
```

A snapshot dump is taken in element LARK. Line 2 gives the parameters for the dump. The instruction at address 010, under location counter 1 is the location where the snapshot request is placed. The address 012, under location counter 1 is the starting address of the dump. Sixteen locations in main storage are dumped along with the contents of the A, X and R registers. The dump is to be taken a maximum of 128 times, but only once every eighth reference.

Example 2:

```
1.  SNAP          JACK
2.  0132,02       HAH+2      0256,4
```

A snapshot is to be taken in element JACK. Line 2 specifies that the instruction at location 0132 under location counter 2 is the location where the snapshot request is placed. The address HAH+2 (where HAH must be externally defined) is the starting address of the dump. 256 or 0400 main storage locations and the contents of the X registers are to be dumped. Since the times and frequency parameters are not specified, the system assumes a value of 100 for times and 1 for frequency.

2.2.2.11. End of Input (END)

Purpose:

Specifies the end of the source language input for the Collector. The END statement is optional. If not given, the end of Collector source language is indicated by the next control statement.

Format:

```
END
```

Example:

```
@MAP,IL
 SERIES  OF
 COLLECTOR DIRECTIVES
END
```

2.2.2.12. Absolute Element Optimization (MINGAP, MINSIZ)

Purpose:

Enables the user to modify the resultant absolute element so as to minimize the I/O transfer time when the program is loaded for execution.

Format:

```
MINGAP    value
MINSIZ    value
```

Parameter:

value — Specifies any positive integer.

Description:

The program areas created by an assembler RES directive or compiler array declarations are unique in that at collection these areas do not contain meaningful data or instructions. The Collector then has two alternatives when defining these RES areas within the generated absolute element:

1. The area could be zero filled. This has the effect of increasing the size of the absolute element which affects the mass storage space requirements of the element as well as the number of words which must be transferred when the element is brought into main storage for execution.

2. The area could be left void. This alternative decreases the size of the absolute element at the expense of increasing the number of access control words (ACWs) and hence the number of I/O operations needed to transfer the element to main storage for execution.

The Collector uses a combination of these two alternatives depending upon the size of the area. Any area within the absolute element greater than or equal to MINGAP words is left void while those less than MINGAP words are zero filled. Each individual ACW required to transfer the element to main storage also controls a minimum of MINSIZ words. Both MINGAP and MINSIZ are initially set equal to 10.

While the value 10 is felt to be optimum in most cases and it generally does not need to be changed, there may be instances, depending upon type of mass storage and program application, where it is desirable to modify the parameters. For instance, increasing the number of words controlled by each ACW, and decreasing the number of I/O operations needed to transfer the program to main storage may reduce the time required to load the program.

2.2.2.13. Program Parameter Specification (TYPE)

Purpose:

Enables the user to specify certain program applicable conditions.

Format:

```
TYPE  parameter-1,parameter-2,...
```

Parameter:

parameter-n may be any of the following:

SETAFCM	Set Arithmetic Fault Compatibility mode for the absolute or relocatable element.
CLRAFCM	Set Arithmetic Fault Non-interrupt mode for the absolute or relocatable element.
INSAFCM	Set the absolute or relocatable element insensitive to the above Arithmetic Fault modes.
EXTDIAG	Produce extended diagnostic tables for the absolute element.

REALTIME — All initial loads and reloads of absolute elements of type REALTIME are done at real-time request priority with real-time positioning. This does not affect processor switching priority.

BLOCKSIZE64 — Set bank sizes into the absolute element in 64 word blocks.

COMSEG — Include all text for a common block in the segment the common block is attached to.

IBJLNK <eltname>,<address>

<eltname> is the name of an element included in the collection, explicitly or implicitly.

NOTE:

Filename may not be used.

<address> is a symbol or numeric value representing an address in the absolute element.

Extend use of BDICALL$/IBJ$ feature to allow collection time definition of parameters to be passed to a routine (see 2.2.9.1).

These parameters may appear in any order on the TYPE directive.

Description:

Within the absolute element's diagnostic tables, the normal entry point name table contains only referenced entry points. The absolute value table contains only referenced absolute entry points and does not include any absolute entry points defined in an element named ERU$ or CERU$. The specification of EXTDIAG prevents any of these exclusions so that all entry points in the element are included in these diagnostic tables.

Parameters SETAFCM, CLRAFCM, and INSAFCM override any related indicators present in the individual relocatable elements included in the collection.

SETAFCM, CLRAFCM, and INSAFCM are only meaningful in collections which produce elements to be executed on an 1110 or 1100/40 System.

The parameter, REALTIME, is used to designate an absolute element as real-time for storage request purposes. All storage requests for programs so designated will be processed at real-time storage priority. The switching level and real-time activity count of the program are not affected until the program does an ER RT$. The storage priority of in-core banks of the program will always be real-time unless all of the program's activities are non-real-time and in a long-wait state, in which case the in-storage banks will be marked long-wait and will be swapable.

Location counters from many elements may define text to be included in a common block. These location counters may be placed, due to rules of element inclusion, in different segments. The segment at the common point of the others contains the actual area to be occupied by the common block. The loading of one of the segments containing text for the common block will cause reinitialization of part of the common block. The presence of the COMSEG parameter will cause all location counters defining text for a common block to be included in the same segment as the common block area, thus avoiding any reinitialization of the common block unless that segment is reloaded.

2.2.2.13.1. Absolute Element Arithmetic Fault Mode Determination

The Collector will mark the Arithmetic Fault mode of an absolute element or Collector produced relocatable element in the following order of precedence:

1. If explicit sensitivity is given on the TYPE statement, the output element is marked accordingly, regardless of the sensitivities of the input relocatable elements.

2. If all input relocatable elements have the same sensitivity, the output element is marked with the sensitivity.

3. If both SETAFCM and CLRAFCM relocatable elements are present, the output element is marked with the sensitivity of the relocatable element containing the program start address; if that element is marked INSAFCM or if no starting address exists, then the output element is marked UNKNOWN. In this case (both SETAFCM and CLRAFCM relocatable elements present), the presence or absence of INSAFCM or sensitivity UNKNOWN relocatable elements is irrelevant.

4. If only SETAFCM relocatable elements are present in addition to INSAFCM and/or sensitivity UNKNOWN relocatable elements, the output element is marked SETAFCM.

5. If only CLRAFCM relocatable elements are present in addition to INSAFCM and/or sensitivity UNKNOWN relocatable elements, the output element is marked CLRAFCM.

6. If INSAFCM and sensitivity UNKNOWN relocatable elements only are present, the output element is marked UNKNOWN.

2.2.2.13.2. EXEC Action Produced by Absolute Element Arithmetic Fault Mode

The Arithmetic Fault mode sensitivity of the absolute element is used by the Executive in determining the initial setting of PSR bit D20. See SPERRY UNIVAC 1110, 1100/40 System Processor and Storage Programmer Reference, UP-7970 (current version). The following describes the action taken by the Executive:

1. If the absolute element's sensitivity is UNKNOWN, the system standard set at system generation is used.

2. If the absolute element's sensitivity is SETAFCM, D20 is initially set.

3. If the absolute element's sensitivity is CLRAFCM, D20 is initially cleared.

4. If the absolute element's sensitivity is INSAFCM ,D20 is initially cleared.

For standard contingency action related to Arithmetic Faults, see Volume 2–Table 4–2.

2.2.2.13.3. Blocksize

When BLOCKSIZE64 is specified on the TYPE statement, all banks of the program will have their sizes stored in the Bank Load Table of the absolute element in 64–word blocks. In the absence of the BLOCKSIZE64 specification, sizes will be stored in 512–word blocks. The operating system will correctly handle either case for any 1100 Series machine, but in the case of the 1100/80 System the presence of BLOCKSIZE64 will improve main storage usage.

2.2.2.14. Program Segmentation (SEG)

Purpose:

Informs the Collector of the beginning of a program segment. All parameters on the SEG directive are optional, except name-1.

Format:

```
SEG  name-1,seg-list
```

Parameters:

name-1	Specifies the name of the segment, name-1.
seg-list	Specifies the address relationship between the segment named in name-1 and the other program segments named in seg-list.

Description:

When name-1 is followed by an asterisk (*), the named segment is automatically loaded when referenced. The asterisk is allowed on all SEG directives, but is ignored if the directive defines the main segment.

The seg-list parameter has several formats which determine the addresses of the segment named in name-1 as follows:

	When seg-list is void, the starting address of the name-1 segment immediately follows the last address of the segment named on last preceding SEG directive.
name-2	Specifies that the starting address of the name-1 segment is the same as the starting address of the name-2 segment. These two segments can never exist in main storage at the same time.
(name-2)	Specifies that the starting address of the name-1 segment immediately follows the last address of the name-2 segment specified.
(name-2,name-3,..., name-n)	Specifies that the starting address of the name-1 segment immediately follows the highest last I-bank and D-bank address of the segments specified in name-2, name-3, name-4, etc. Note that the highest last I-bank address may be contained in a segment different than the one containing the highest last D-bank address.
( )	Specifies that the starting address of name-1 segment immediately follows the highest last address of all segments previously named.

For additional information of the SEG directive, see 2.2.4.2.

2.2.2.15. Relocatable Segments (RSEG)

Purpose:

Specifies the named segment as a relocatable segment. A relocatable segment (RSEG) is one whose location within the program is determined dynamically by the program during execution rather than at collection. An RSEG may reference entry points within the program, but the RSEG itself may not contain definitions for references elsewhere in the program (unless the reference is of the form J TAG,x where x contains the address at which the RSEG was loaded) since only internal RSEG addressing is relocated during segment loading.

Format:

```
RSEG    name
```

Parameter:

name — Specifies the relocatable segment.

Description:

For further information on relocatable segments, see 2.2.4.3.

2.2.2.16. Dynamic Segments (DSEG)

Purpose:

Provides a mechanism by which the program area occupied by a segment will not be included in the initial program requirement for main storage.

Format:

```
DSEG    name-1,seg-list
```

Parameters:

name–1	Specifies the name of the segment.
seg–list	Specifies the address relationship between the segment named in name–1 and the other program segments (see 2.2.2.14).

Description:

Dynamic segments are identical to normal overlay segments (defined by the SEG directives – see 2.2.2.14) in all aspects except one: the program area assigned exclusively to dynamic segments is not included when determining initial program size. It is the programmer's responsibility to guarantee that the program area is available by using the MCORE$ request (see Volume 2–4.7.1) prior to requesting segment loading or to request segment loading via the SYS$*RLIB$ routine DLOAD$, which will obtain the necessary program area prior to initiating the segment load (see 2.2.4.5.4).

2.2.2.17. Executive Function Arrangement (XSEG)

Purpose:

Allows the Collector to place segments contiguous to one another within main storage blocks, without wasted storage gaps between them. This statement is primarily intended for use in collections of the EXEC system.

All parameters on the XSEG statement are optional, except name–1.

Format:

```
XSEG    name–1,seg–list
```

Parameters:

name–1	Specifies the name of the segment.
seg–list	Specifies the address relationship between the segment named in name–1 and the other program segments. (See 2.2.2.14 for seg–list formats.)

Description:

An XSEG directive functions the same as a SEG directive, except that the initial relative starting address is reduced, if possible, by multiples of 01000. The resulting final starting address is equal to the starting address of the bank which the segment is in, plus the number of words by which the initial relative starting address exceeds a multiple of 01000.

Example:

```
    @MAP,I      TEST,TEST
    I-BANK      XS1,02000
1.  SEG      MAIN
     IN     EL1
2.  XSEG      A
     IN   EL2
3.  XSEG     B
     IN     EL3
4.  SEG      C,B
     IN     EL4
5.  XSEG     D,(B,C)
     IN     EL5
```

Assume the following segment lengths:

Segment	Octal Length
MAIN	01143
A	02054
B	02712
C	04364
D	0115

The following shows the assigned addresses for the segments:

Segment	Assigned Address Range
MAIN	02000 - 03142
A	02143 - 04216
B	02217 - 05130
C	02217 - 06602
D	02603 - 02717

1. Segment MAIN is assigned a start address of 02000 (see 2.2.2.14 for bank start address assignment), and a last address of 03142.

2. Segment A, an XSEG which follows segment MAIN, starts at address 02143 and ends at address 04216. Segment A is initially assigned a start address of 03143, immediately following segment MAIN. However, since segment A is an XSEG, its start address of 02143 is formed by adding 0143 (the number of words which 03143 is over a multiple of 01000) to 02000 (the start address of bank XS1).

3. Segment B, an XSEG which follows segment A, starts at address 02217 and ends at address 05130. Segment B is initially assigned a start address of 04217, immediately following segment A. Since segment B like segment A is an XSEG, its actual start address is 02217 (0217 + 02000).

4. Segment C starts at the same address as segment B with a start address of 02217 and last address of 06602.

5. Segment D is an XSEG which follows the highest address assigned to segment B and C. The initial start address is 06603 immediately following segment C. The actual start address for segment D is 02603 (0603 + 02000), since it is also an XSEG.

2.2.2.18. Bank Structuring (IBANK and DBANK)

Purpose:

Specifies the beginning of a program bank within a bank-named collection.

All parameters on the IBANK and DBANK directives are optional, except name1.

Format:

```
IBANK,options     name-1,bank-list
DBANK,options     name-1,bank-list
```

Parameters:

options — See Table 2-2.

name-1 — Specifies the name of the bank.

bank-list — Specifies the address relationship between the bank named in name-1 and other program banks.

Description:

Names present in a bank-list may be I-bank and/or D-bank names. The bank-list parameter has several formats which determine the location of the bank named in name-1 as follows:

When bank-list is void, the starting address of the name-1 bank is the next address which is a multiple of 01000 following the most recently defined IBANK (if name-1 is an IBANK) or DBANK (if name-1 is a DBANK).

name-2 — The starting address of the name-1 bank is the same as that of the name-2 parameter. Name-2 may be either a bank name, a numeric value (octal or decimal), or a bank name ± offset, where offset is an octal or decimal numeric value which is added to or subtracted from the start address of the bank name.

(name-2) The starting address of the name-1 bank is the next address which is a multiple of 01000 following the name-2 bank. Name-2 may be a bank name or a bank name ± offset, where offset is applied to the bank's last address.

(name-2, name-3,..., name-n) The starting address of the name-1 bank is the next address which is a multiple of 01000 following the highest of the name-2,...,name-n banks. Each of the name-2,...,name-n parameters may be a bank name, a bank name ± offset, or an octal or decimal value.

For additional information on bank-named collections, see 2.2.5.

Example:

```
1.  IBANK    I1
2.  IBANK    I2
3.  DBANK    D1
4.  IBANK    I3,I2
5.  DBANK    D2,036000
6.  DBANK    D3,(I2,D1)
7.  DBANK    D4,(I3)
8.  IBANK    I4,I2+03000
9.  DBANK    D5,(D3+02000,D4,D2)
```

The program is divided into four I-banks and five D-banks.

1. I1 starts at address 01000.

2. I2 starts at the next 01000 following I1.

3. D1 starts at 040000 or at the next 01000 following the longest I-bank, whichever address is higher.

4. I3 starts at the same address as I2.

5. D2 starts at address 036000.

6. D3 starts at the next 01000 following the higher last address of I2 and D1.

7. D4 starts at the next 01000 following I3.

8. I4 starts at the address obtained by adding 03000 to the start address of I2.

9. D5 starts at the next 01000 following the highest last address of D4, D2, and start address of D3+02000.

Table 2-2. IBANK and DBANK Directive, Options

Option Character	Description
C	This bank is to be the control bank. This option may only be specified once in a collection. If no C option appears in a collection, the control bank will be selected from the existing Main D–, Utility D–, Main I–, and Utility I–banks, in that order.
D	This bank is a dynamic bank. If the D option is not specified, the bank is assumed to be a static bank.
E	This bank prefers to be loaded into extended storage (on 1110, 1100/40 only).
$E	This bank must be loaded into extended storage.
H	This bank requires common data bank contingency handling.
L	References to BDICALL$ and IBJ$, where the associated reference is defined in this bank, are to be satisfied by 0 and LMJ, respectively. This option is only allowed on static initially based banks, and is assumed for the control bank (see 2.2.9.1).
M	This bank is to be initially based on the Main PSR. This option may be used only once with an IBANK directive and once with a DBANK directive. If unspecified, the M is assumed for the first IBANK and first DBANK directives. Once the M option is specified, no assumptions are made regarding the M or U option.
P	This bank prefers to be loaded into primary storage (on 1110, 1100/40 only).
$P	This bank must be loaded into primary storage (on 1110, 1100/40 only).
Q	This bank will use TS queueing.
R	This bank is to be read only (write–protected).
S	This bank, while included in the absolute code, is treated by the system similarly to a common bank. The bank is treated by the Collector like a dynamic unbased bank (i.e. D option with no M or U option). This option can be used to test common banks without the need to load the bank into the system library. Segmentation is not allowed in banks with the 'S' option, and if the S option is used on the control bank, no segmentation is allowed in the program.
U	This bank is to be initially based on the Utility PSR. This option may be used only once with an IBANK directive and once with a DBANK directive. The U option is never assumed on an IBANK or DBANK directive (on 1110, 1100/40 only).

Table 2-2. IBANK and DBANK Directive, Options (continued)

Option Character	Description
V	Assign all addresses, but strip off this bank's code. Can be used to strip individual bank's code rather than all I-bank or all D-bank code as is done by the V and Y options on the @MAP control statement.
X	This is a common bank (used only for initial basing). This option must be used in conjunction with the M or U options only. No SEG, IN or FORM directives may follow this BANK directive.

2.2.2.19. Location Counter Set Specification ($lcs)

Purpose:

Specifies the set of location counters to be included in the current bank from elements named, or present in all relocatable elements in whole files named for inclusion.

Format:

$lcs

Parameter:

$lcs — Specifies the set of location counters either via a keyword or explicit naming as follows:

$ALL	include all location counters.
$NONE	include no location counters (used to create dummy or skeleton structures).
$ODD	include only odd location counters.
$EVEN	include only even location counters.
$$n_1,n_2,...,n_m$$	include only those location counters specified.
$ALLBUT,$$n_1,n_2,...,n_m$$	include all location counters except those specified.

Description:

The location counter set specification is used only in conjunction with a bank-named collection. The specified set remains in effect until the next $lcs or IBANK or DBANK directive is encountered. A bank statement automatically sets the location counter set to $ODD for IBANK and $EVEN for DBANK. The location counter set specified for a bank may be individually overridden for individual elements or files by using the optional $lcs field on the IN directive (see 2.2.2.1).

2.2.2.20. Source Language Structure Duplication (FORM)

Purpose:

Allows the duplications of a portion of a program structure previously defined within a map without requiring repetition of the source language used to define that structure.

Format:

```
FORM    bank-name
FORM    bank-name*seg-name
```

Parameter:

bank-name	Specifies the bank whose structure is to be duplicated.

seg-name — Specifies the segment within the bank whose element inclusion structure is to be duplicated.

Description:

The FORM is useful where a bank or segment structure is to be duplicated and only a location counter change is desired.

When placed following an I-BANK or D-BANK statement a 'FORM bank-name' directive causes the full segment tree of the named bank to be regenerated for the bank being defined, except that the location counter set presently in control must differ from the one which was in control for the bank named on the FORM directive. If an explicit location counter set is specified, it must precede the FORM statement. Since 'FORM bank-name' creates an exact duplication of a previous bank structure, no additional SEG, DSEG, RSEG, or IN directives may be specified for the bank being generated.

A 'FORM bank-name*seg-name' directive causes the IN directives within the specified segment to be regenerated.

Only SEG, DSEG, and IN directives within a structure are duplicated when a FORM on that structure is specified.

Examples:

```
I-BANK     I1
SEG      MAIN
IN       A,B
SEG       SG
IN       FILE.ELT,.ELT1
D-BANK     D1
SEG       MAIN
IN       A,B
SEG      SG
IN     FILE.ELT,.ELT1
```

The use of the 'FORM bank-name' directive can be used as follows to produce the same results as the above example:

```
I-BANK      I1
SEG       MAIN
IN       A,B
SEG       SG
IN       FILE.ELT,.ELT1
D-BANK      D1
FORM      I1
```

As shown below, the 'FORM bank-name*seg-name' directive can be used to produce the same results as the two preceding examples:

```
I-BANK      I1
SEG       MAIN
IN       A,B
SEG        SG
IN      FILE.ELT,.ELT1
D-BANK      D1
SEG       MAIN
FORM       I1*MAIN
SEG        SG
FORM       I1*SG
```

2.2.3. Functional Aspects of the Collector

After the Collector has interpreted the parameters of the @MAP control statement (see 2.2.1) and the parameters of the Collector directives (see 2.2.2), there remains the combining of the relocatable elements into a relocatable or absolute element and the insertion of the final element into the program file to complete the collection process.

2.2.3.1. Collector-Produced Relocatable Elements

Although the Collector is generally used to produce an absolute element, a relocatable element can be produced by specifying the R option on the @MAP control statement. All indicated relocatable elements are merged into a single element and only the external definitions specified in the DEF directive are retained. All other external definitions are submerged in the new relocatable element.

The R option is used most often when the user wants to include a relocatable element more than once in an absolute element. Initially, an R option collection is performed. This combines the desired element with a specified set of relocatable elements. As long as no external definitions within the element are specified on the DEF directive, the desired element is submerged into the newly-created relocatable element. The original relocatable element and the newly-created relocatable element in which it has been submerged can both be collected in a single absolute element. Relocatable elements in SYS$*RLIB$ are not implicitly included in an R option collection. Location counter specifiction is preserved, i.e., information contained in location counter n for all input RB's is placed under location counter n of the output RB.

Only the following directives may be specified with an R option collection. All others produce a diagnostic message and the collection continues.

```
TYPE    DEF
ENT     IN
LIB     NOT
REF     CLASS
END     EQU
```

2.2.3.2. Element Inclusion

Adding elements to a collection is a two-part process:

1. Finding the files that were specified in the Collector directives (see 2.2.2).

2. Finding within these files the elements that have been specifically named on IN directives (see 2.2.2.1) or that contain entry points which satisfy the undefined symbols.

Elements to be included in the collection may have been specified on IN directives (see 2.2.2.1) either with or without the filenames in which those elements appear. For those elements with a filename present on the IN directive, the Collector immediately references that file, finds the element, and processes the preamble of the element. After the preamble of a relocatable element has been processed, the text (instructions and data) of the relocatable element becomes part of the final output element.

If a @PREP (see 4.2.11) of TPF$. has not occurred, all relocatable elements in TPF$. are tentatively included in the collection unless specifically excluded via the NOT directive (see 2.2.2.2). After all elements for a collection have been found, any nonreferenced element that was tentatively included from TPF$ is eliminated from the collection.

If a @PREP of TPF$. did occur, an element from TPF$, is included only if it is named on an IN directive or if one of its external definitions satisfies an undefined reference from another element included in the collection. When a @PREP of TPF$. has occurred, TPF$. is always the first file searched when attempting to locate elements named without a filename and elements with external definitions satisfying undefined references.

The situation may arise where the user wants to implicitly include elements from a file other than TPF$, having entry point names or element names which duplicate entry point or element names in TPF$. Since TPF$ is searched prior to searching other files, the elements from TPF$. are included instead of the desired elements which are in other files.

Therefore, if duplicates of element names and external definitions are present and those in TPF$. are not wanted in the collection, a @PREP of TPF$. is needed to prevent automatic inclusion of the TPF$. elements. It is also necessary to specifically IN by filename and element name any elements from other files which have element names or external definitions duplicated by TPF$. elements.

For those elements without a filename on the IN directive, the Collector searches files for these elements in the following order:

1. The Temporary Program File (TPF$)

2. User-defined files that were indicated by the LIB directive (see 2.2.2.3)

3. The System Relocatable Library (SYS$*RLIB$)

In an attempt to satisfy all undefined references in the collection, the Collector searches the specified files for elements that have entry point names that correspond to the symbol names appearing in the Collector created UNDE table (see 2.2.3.3). The UNDE table contains all the symbols that are present in the undefined symbol tables of the processed element preambles. When an element is found with an entry point name corresponding to a symbol name in the UNDE table, the preamble of that element is processed and the now defined symbol is removed from the UNDE table. The order of search for undefined symbols is:

1. The Temporary Program File (TPF$)

2. User-defined files defined by the LIB directive (see 2.2.2.3) and which were previously @PREPed

3. The System Relocatable Library (SYS$*RLIB$)

In a bank-implied collection, the included elements are placed in the instruction and data areas of the final absolute element. Odd numbered location counters of an element are assigned to the instruction area. Even numbered location counters and common blocks are assigned to the data area. See 2.2.2.1 and 2.2.4.6 for information on specific placement of common blocks.

See 2.2.5.6 and 2.2.5.7 for determining element inclusion and placement in a bank-named collection.

The most efficient collection results when every element desired in the collection is explicitly named, including filenames; this eliminates @PREP requirements and library searches.

2.2.3.3. Processing Element Preambles

An element preamble is attached to every relocatable element created by the language processors and the Collector. The element preamble provides information which is needed when collecting relocatable elements to form an absolute or relocatable element. This information includes:

- The definition and location of each externally-defined name (entry point) in the element
- The length in words, under each location counter in the element
- The table of the undefined symbols appearing in the element
- Common blocks in the element

When the preamble is processed:

- The entry points in the element are added to the Collector entry point table (EP table).
- Undefined symbols appearing in the element which have no corresponding entry in the EP table are listed in the UNDE table.
- Undefined symbols in the element which have a corresponding name to an entry point in another element are linked to the EP table.

Symbols are removed from the UNDE table as corresponding entry point names are found. Newly encountered undefined symbol names are added at the end of the UNDE table.

2.2.3.4. Instruction and Data Area

This section applies primarily to bank-implied collections. See 2.2.5 for bank-named collection considerations.

Every program containing segments in addition to the main segment always has a D-bank. If there is no program data in the D-bank, then it contains at least a segment load table. The segment load table contains an entry with information describing every segment of the program. It is located preceding the user's main segment D-bank. Since the segment load table has no main storage protection, special care has to be taken not to destroy its information.

The program's data, when it exists, is located after the segment load table and any other Collector-produced tables, such as the ENTRY$, COMMN$, XREF$, and indirect load table.

The first address of the I-bank (instruction area) is assigned 01000. The starting address of the D-bank (data area) is dependent upon the size of the I-bank, the possible use of the assembler SETMIN directive, the total program size, and the options specified on the @MAP control statement. The first

address of the D-bank, however, is always a multiple of 01000 and is usually given the value 040000.

Odd numbered location counters are assigned to the I-bank; even numbered location counters and common blocks are assigned to the D-bank.

The user program can reference the first and last I-bank and the first and last D-bank addresses by the symbols: FRSTI$, LASTI$, FRSTD$, and LASTD$, respectively. The Collector replaces these symbols with the actual assigned address values.

An unnamed common block (i.e., blank common), if required in the program, is attached to the main segment under the name BLANK$COMMON. It may be positioned in another segment by an IN BLANK$COMMON directive.

Named common blocks (if not named in an IN directive) are attached to the segment which is located at the common point in the paths to the main segment of all segments referencing it.

2.2.3.5. Collecting Reentrant Processors

In creating reentrant processors it is not only more efficient to explicitly name all elements including their filenames but it is also extremely advisable. The nature of reentrant processors dictates that from one collection to another all elements be located in the same relative position within the absolute element. This can only be ensured by explicitly including all elements in every collection of the absolute element. (See 2.4 for reentrant processor preparation.)

2.2.4. Program Segmentation

When an absolute program is being executed, it must reside in main storage. However, there may not be enough available area in main storage to contain the complete program. Therefore, the program must be subdivided or segmented so that the various parts or segments can be loaded into main storage as the program is being executed.

Even when the total program size may fit into main storage, many times it is advantageous to subdivide the program into functionally independent units (segments) which are loaded into main storage only when needed. This reduction in the program's main storage requirements reduces main storage impact while allowing increased storage utilization.

Another way of dividing a program is by specifying banks as logical units. Banks may be used in conjunction with or independent of program segmentation (see 2.2.5).

A segmented program consists of:

- one main segment which resides in main storage throughout the execution of the program, and
- subordinate segments which are loaded into main storage as they are needed.

As each subordinate segment is loaded into main storage, it may overlay all or part of a previously loaded segment. The area overlayed is equal in size to the size of the new segment. The main segment is never allowed to be overlayed except by a relocatable segment (RSEG).

The absolute element resulting from the collecting of various relocatable elements may or may not be segmented. However, a nonsegmented program can be functionally considered a segmented program with only a main segment.

2.2.4.1. Segmentation Directives

The directives needed to specify program segmentation are as follows:

- SEG – Informs the Collector of the beginning of a segment (see 2.2.2.14).
- RESG – Informs the Collector of the beginning of a relocatable segment (see 2.2.2.15).
- DSEG – Informs the Collector of the beginning of a dynamic segment (see 2.2.2.16).
- IN – Specifies the elements to be included in the segment (see 2.2.2.1).
- NOT – Specifies which elements are to be excluded (see 2.2.2.2).

Segments may be loaded and executed independently; however, elements common to several segments must be in main storage when the referencing segments are executed.

Since each segment has a path leading back to the main segment (defined by the relationships specified on the segmentation directives), elements which are implicitly included and which are referenced by two or more segments are attached to the segment which is located at the common point in the paths to the main segment of all referencing segments. Elements specified on IN directives are never moved from the segment in which they were specifically placed.

2.2.4.2. SEG Directive Considerations

The IN directive specifies the files and elements to be included in a segment. If no SEG directive is encountered prior to the first IN directive following the @MAP control statement, a SEG $MAIN$ directive is assumed by the Collector and it applies to all following directives until a SEG, RSEG, or DSEG directive is encountered.

The segment name contains 1 to 12 alphanumeric characters (with the $ and – allowed) in length. The segment name must not be the same as any entry point name in the collection, and must contain at least one alpha character.

Within a segment, any elements included to satisfy undefined symbols are located at the beginning of the segment in the inverse order of their inclusion; that is, the last included element is the first element in the segment. Following any implicitly included elements are those named on IN directives in the exact order they were named. When all elements within a file are included in a segment, by specifying only the filename on the IN directive, the ordering of the file's elements is random.

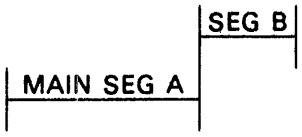
Example 1:

```
1. SEG A
2. SEG B
3. SEG C, (A)
```

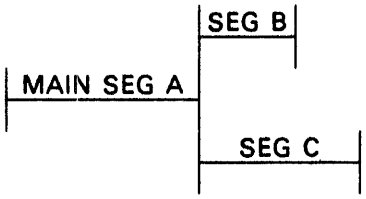
The program is to be divided into three segments.

1. Specifies segment A as the main segment.

2. Specifies that the starting address of segment B is immediately after the last address of main segment A.



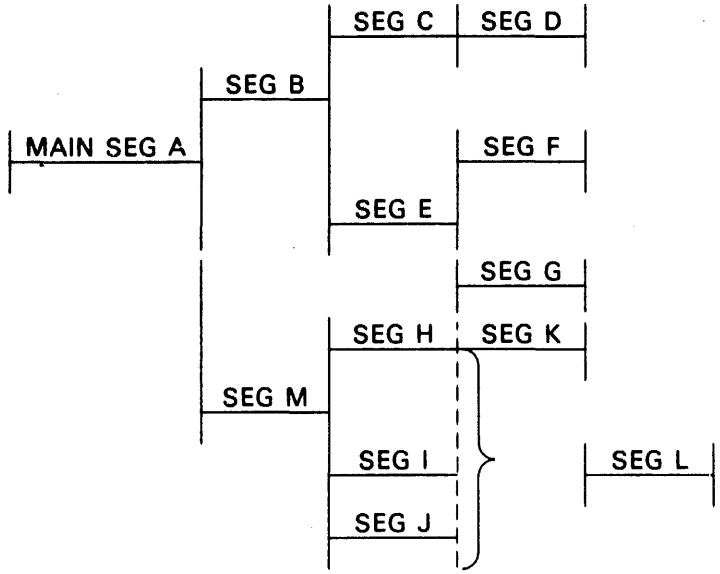
3. Specifies that the starting address of segment C is immediately after the last address of main segment A. Segments B and C can never exist in main storage at the same time.



Example 2:

```
1.  SEG  A
2.  SEG  B
3.  SEG  C
4.  SEG  D, (C)
5.  SEG  E,  C
6.  SEG  F, (E)
7.  SEG  G,  F
8.  SEG  M,  B
9.  SEG  H, (M)
10. SEG  I,  H
11. SEG  J,  H
12. SEG  K, (H,I,J)
13. SEG  L, ()
```

The program is to be divided into thirteen segments.



1. Specifies segment A as the main segment.

2. Specifies that the starting address of segment B is immediately after the last address of main segment A.

3. Specifies that the starting address of segment C is immediately after the last address of segment B.

4. Specifies that the starting address of segment D is immediately after the last address of segment C.

5. Specifies that the starting address of segment E is the same as the starting address of segment C.

6. Specifies that the starting address of segment F is immediately after the last address of segment E.

7. Specifies that the starting address of segment G is the same as the starting address of segment F.

8. Specifies that the starting address of segment M is the same as the starting address of segment B.

9. Specifies that the starting address of segment H is immediately after the last address of segment M.

10. Specifies that the starting address of segment I is the same as the starting address of segment H.

11. Specifies that the starting address of segment J is the same as the starting address of segment H.

12. Specifies that the starting address of segment K is immediately after the last address of either segment H, I, or J, whichever segment is longest.

13. Specifies that the starting address of segment L is immediately after the highest last address of all segments in the set: A, B, C, D, E, F, G, H, I, J, K, and M.

2.2.4.3. RSEG Directive Considerations

The elements included in the relocatable segment should be explicitly named on IN directives (see 2.2.2.1). When an element which is referenced by more than one segment is implicitly included it is placed in a segment other than the RSEG. Generally, it is advisable if an element is referenced by more than one segment (one of which is an RSEG), that the element be explicitly included in the main segment.

Relocatable segments may not be indirectly loaded. See 2.2.4.5.1 for the direct method of loading segments.

The starting address of a relocatable segment has no relationship to other segments in the collection. An RSEG may be loaded at whatever starting address is given in register A2 during the LOAD$ calling sequence (see 2.2.4.5.1). The LOAD$ request adds the value in register A2 to all relative address references internal to the named relocatable segment. Any references to RSEG labels from outside the relocatable segment must be user-modified by the value in register A2.

All instructions and data in a relocatable segment are collected together with all odd location counters followed by all even location counters.

A relocatable segment may be loaded into either an I- or D-bank of the program.

An element within an RSEG can define a common block which is located outside that RSEG only if the common block of the RSEG element contains no text. This should be especially noted by users who employ higher level languages such as FORTRAN, JOVIAL, etc.

Example:

```
1.  RSEG  ORSON
2.  IN    ORSON
```

Element ORSON is specified for inclusion in the relocatable segment ORSON.

2.2.4.4. DSEG Directive Considerations

Dynamic segments may be defined as requiring indirect load, and may be placed anywhere within the segment structure of a program.

The program addresses assigned to DSEGs are not considered when determining the initial I- and D-bank limits for a program. However, in assigning the values in LASTD$ and LASTI$ (see 2.2.9), DSEG addresses are used.

2.2.4.5. Loading Program Segments

When a segmented program is called for execution (see 2.3.1), only the main segment is initially loaded. The subordinate segments are loaded by either

- the direct method, or
- the indirect method.

Whenever a segment is loaded, an initial copy of the segment is loaded. Once loaded, a segment will remain marked as loaded until all or part of its main storage space is overlayed by another segment or released via ER LCORE$ (see Volume 2–4.7.2).

See 2.2.5.8 for additional information on loading program segments in bank-named collections.

2.2.4.5.1. Direct Method (L$OAD and LOAD$)

When using the direct method of loading, use either

- the L$OAD procedure, or
- the Executive Request LOAD$.

Format of the L$OAD procedure:

```
L$OAD   name,jump,clear,rseg-addr,bank-name
```

Parameters:

name	Specifies the name of the segment to be loaded.
jump	Specifies the location where control is to be transferred after the segment is loaded; if omitted, control passes to the location following the LOAD$ request.
clear	If greater than zero, the program area containing the segment to be loaded is not zero filled prior to segment load. If zero, the area to be occupied by the segment is zero filled prior to segment load.
rseg-addr	If the segment was defined by an RSEG directive, this parameter specifies the starting address for the relocatable segment. If omitted when loading a relocatable segment, the address must be in register A2 before the call is made: `L,U  A2,rseg-address` The address may be defined as an octal value or a tag not contained in an RSEG.
bank-name	Used only for loading an RSEG within a bank-named collection. Specifies the bank into which the RSEG is to be loaded. If omitted, the bank-name must be in register A2 along with the RSEG starting address: `LXI,U  A2,bank-name`

Description:

The L$OAD procedure produces a sequence of code which loads: register A0 with the segment name, register A1 with jump address, register A2 with bank name and address for RSEG, and generates the LOAD$ request. The LOAD$ request takes the form:

```
L,U  A0,seg-name or L A0,(0400000,seg-name)
L,U  A1,jump
L,U  A2,rseg-addr or L A2,(bank-name,rseg-addr)
ER   LOAD$
```

Seg-name is the same as the contents of the name-1 parameter in the SEG directive (see 2.2.2.14).

When bit 35 of register A0 is set, the segment loader does not clear the main storage area to be occupied by the segment. This decreases the time required to load the segment, but as a result, any areas within the segment that are not initialized with data and instructions cannot be assumed to be zero.

Examples:

```
1.  L$OAD   NEW,ORG1
2.  L$OAD   CAP,YELL,0,01350
```

1. After segment NEW is loaded, transfer control to location ORG1. The area occupied by segment NEW is zero filled prior to loading. The L$OAD procedure produces the same effect as the code:

```
L,U   A1,ORG1
L,U   A0,NEW
ER    LOAD$
```

2. The area to be occupied by relocatable segment CAP is zero filled prior to loading. The starting address for segment CAP is 1350_8. After the segment is loaded, control is passed to YELL. The L$OAD procedure produces the same effect as the code:

```
L,U   A2,01350
L,U   A1,YELL
L,U   A0,CAP
ER    LOAD$
```

2.2.4.5.2. Indirect Method

Whenever a segment that is marked for indirect loading is referenced by any jump instruction that passes control to the segment's I-bank area, the segment is automatically loaded if it is not already in main storage. Segments to be loaded by the indirect method must be so marked on the SEG directive. The mechanics for such loading are set up by the Collector and carried out by the segment loader. The Collector replaces the address portion of the jump command with the address of an indirect load table entry. The indirect load table performs an SLJ instruction to the indirect load routine which, in turn, performs an ER to the segment loader (if the segment is not already loaded) and jumps to the location of the externally defined symbol. All registers are preserved by the process. The indirect load table is assigned to the data area of the main segment.

If indirect loading is used, the reference may not be made to an external symbol with an offset.

If the B option was specified on the @MAP control statement, the indirect load routine indicates that the segment's main storage area need not be zero filled.

Segments marked for indirect loading may also be loaded by the direct method.

No instruction interpretation is done to ensure that a referencing instruction is in fact a jump instruction.

The indirect load routine is nonreentrant.

Example:

```
SEG    EAP*
SEG    NINE*,(FR,SX)
SEG    SAL*,(PEN)
```

Segments EAP,NINE, and SAL are automatically loaded when any externalized I-bank entry point is referenced.

2.2.4.5.3. Reloading the Main Segment

It may be desirable to re-establish the main segment of a program for either error recovery or reinitialization. This is done by the LOAD$ request. The LOAD$ request reloads the entire main segment including all initially-produced collector tables. The main storage requirements remain unchanged.

The calling sequence is:

```
L     A0,(clear,0400000)
L,U   A1,reentry-addr
ER    LOAD$
```

The first coding line loads register A0 with the segment-id of 0400000. The clear parameter functions as follows:

- If clear is less than 0, the main segment area is not cleared before loading.
- If clear is greater than or equal to 0, the main segment area is cleared.

The second coding line loads register A1 with the reentry-address according to the following:

- If the reentry-addr is equal to 0, control is returned to the instruction following the LOAD$ request.
- If the reentry-addr is not zero, control is passed to that address.

See 2.2.5.8.3 for additional information regarding main segment reload in bank-named programs.

2.2.4.5.4. Loading Dynamic Segments (D$LOAD and DLOAD$)

Dynamic segments may be loaded by referencing the dynamic load routine contained in SYS$*RLIB$. This routine may be referenced explicitly by the user or may be referenced implicitly by the indirect load routine when a DSEG is marked for indirect load. The dynamic load routine may be called for loading either DSEG's or normal segments.

The dynamic load routine will determine whether the segment to be loaded is in fact a dynamic segment. If it is not, an ER LOAD$ is executed for the segment. If the segment is a dynamic segment,

the routine will acquire the necessary DSEG areas via ER MCORE$ and then execute an ER LOAD$ to load the DSEG.

The user may reference the dynamic load routine by using the D$LOAD procedure or by a jump to DLOAD$.

Format:

```
D$LOAD    segname,jump,clear
```

Parameters:

segname	Specifies the name of the segment to be loaded.
jump	Specifies the location where control is to be transferred after the segment is loaded; if control is to be passed to the location following the jump to DLOAD$, a *0 must be placed in the jump parameter field or a tag must be placed following the procedure call and must be specified in the jump parameter field.
clear	If greater than zero, the program area containing the segment to be loaded is not zero filled prior to segment load. If zero, the area to be occupied by the segment is zero filled prior to segment load.

Description:

The D$LOAD procedure generates the following sequence of code:

```
L,U  A0,segname    or   L   A0,(0400000,segname)
L,U  A1,jump
J    DLOAD$
```

When bit 35 of register A0 is set, the segment loader does not clear the main storage area to be occupied by the segment.

2.2.4.5.5. Releasing a Segment's Program Area (D$REL and DREL$)

When the main storage area occupied by a segment is no longer required by the program, the space may be released by referencing the dynamic release routine in SYS$*RLIB$. This routine may be referenced by either the D$REL procedure or by a jump to DREL$.

Format:

```
D$REL  segname,jump
```

Parameters:

segname	Specifies the name of the segment area to be released.
jump	Specifies the location where control is to be transferred after the segment's area is released; if omitted, control passes to the location following the J DREL$ call.

Description:

The D$REL procedure generates the following sequence of code:

```
LXI,U    A1,segname          LXI,U   A1,segname
LXM,U    A1,jump      or     LMJ     A1,DREL$
J        DREL$
```

The dynamic release routine will determine the program I-bank start address of the named segment. This segment may be either a DSEG or overlay segment. This address minus one is placed in A0 and an ER LCORE$ is executed. The D-bank start address minus one is then used for an ER LCORE$. Following the release of the program area, the segment is marked as a dynamic segment. Its area may then be reacquired by referencing DLOAD$.

If the released program areas are occupied by more segments than that specified on the D$REL call, these other segments will not be marked as DSEG's. Therefore, it is the user's responsibility to call DREL$ for each segment actually released or to assure the necessary program areas are reacquired prior to a subsequent reload of any such segment.

2.2.4.6. Use of Common Blocks

The Collector produces, in the absolute element, load control specifications for the LOAD$ routine. These specifications indicate which text words (data and instructions) are to be put at which locations in main storage when the segment is loaded.

If a common block is given initial values (filled with text, rather than simply set aside as a reserved area), the Collector produces specifications to put in these values when the segment containing the element which defines the initial values is loaded.

For example, if five different elements define five different initial values for the same common block and each of these five elements was in a different segment, the same common block located in a segment common to all referencing segments would be reinitialized each time one of the five different segments was loaded. This occurs regardless of where the referenced common block is located within the user's program area.

Any areas of the common blocks in which text is not loaded upon reinitialization are not changed as long as the reinitialization is caused by the loading of a segment other than the one in which the common block resides.

On IN directives (see 2.2.2.1), common blocks are always specified without a filename. A common block name must not be identical to an element name.

If TPF$ is not prepped, the size of any common blocks in TPF$ will be used in determining the final size of the common block in the absolute or relocatable element, otherwise, the largest size declared in any element is used as the size of the common block.

Bank-named collections may not contain locally included common blocks (see 2.2.5.6.2).

2.2.5. Bank-Named Collections

2.2.5.1. General

The Collector provides the capability to subdivide an absolute element into logical structures called banks. This is done by explicitly defining the banks with the Collector's IBANK and DBANK source language directives and then defining the segmentation and element inclusion structures within the bank. In the case where there is no explicit bank directive, the Collector generates a two-bank absolute element, with odd location counters of relocatable elements to the IBANK and even location counters to the DBANK. This is the bank-implied collection.

The definition of a bank is that entity of a program which may be specified by a single EXEC system bank descriptor word.

2.2.5.2. Bank Address Assignments

The relative starting address of a bank may be explicitly specified on the IBANK or DBANK directive (see 2.2.2.18) as a numeric value or as an address to be determined in relation to the size of other banks. When no such address specification exists, the following is applied:

a. The I-bank specified via the first IBANK directive starts at program address 01000.

b. The D-bank specified via the first DBANK directive starts at the higher of:

   1. 040000

   2. The next address which is a multiple of 01000 following the highest IBANK address

c. I-banks follow I-banks and D-banks follow D-banks; i.e., an I-bank (D-bank) which has no explicit address assignment is assigned a starting address which is the next multiple of 01000 following the last address assigned to the previous I-bank (D-bank) defined in the Collector source language.

No I-bank address can be greater than 0177777 (65K) and no D-bank address can be greater than 0777777 (262K). The total size of all I-banks (D-banks) can be greater than 0177777 (0777777) provided there is some overlap of program address assignments within banks, so that the total address space does not exceed 0177777 for I-banks or 0777777 for D-banks.

Overlap of bank address assignments may be achieved by using the IBANK and DBANK directives in a fashion similar to that of the SEG directives. However, whereas overlay segments having the same logical (program) address space also occupy the same physical space, banks of the same address space do not overlay each other and may occupy different areas of physical space simultaneously.

In addition to the collector-defined tags FRSTI$, LASTI$, FRSTD$, and LASTD$ (see 2.2.9), the tags FIRST$, LAST$ and BDI$ are defined for the bank-named collections. These tags are defined to be the first assigned address, last assigned address, and bank descriptor index value for the bank in which the reference is contained.

The SPERRY UNIVAC 1100 Series Assembler SETMIN directive and MASM's $INFO 3 directive allow the user to specify the minimum D-bank address of the element which contains the directive.

2.2.5.3. Initially-Based Banks

An initially-based bank is a bank which is referenceable at the start of execution of a program. This means that the bank is described in one of the bases (B_1 or B_D of either the Main or Utility PSR. A bank may be specified for initial basing via the M or U option on the I-BANK and D-BANK directives (see 2.2.2.18). Each option may be used with a maximum of one I-BANK and one D-BANK directive. When there is no initial basing specified for any banks, then the first I-bank and the first D-bank defined in the Collector source language are initially based on the B_I and B_D, respectively, of the Main PSR. A bank is never assigned to be based on the B_I or B_D of the Utility PSR by default. Thus, if a bank is to be initially based on the Utility PSR, it must have the U option specified on the I-BANK and D-BANK directives. Note that a bank specified for initial basing on the Utility PSR can be executed only on an 1110 or 1100/40 System. If a bank is not initially based, it can be based dynamically via the LBJ, LIJ or LDJ instructions (see 2.3.3.2).

2.2.5.4. The Control Bank

Since any bank may have its basing removed during program execution, it is necessary to designate one bank which, except in unusual circumstances, will not have its basing removed via an LIJ or LDJ to another bank. This bank, known as the control bank, is assumed to be the normal place in which to collect such should-always-be present components of a program as permanent flags, central control routines for sets of dependent relocatable library subroutines, Test and Set cells, and locations designed to capture interrupts (such as guard mode contingency) which can result at varied or unpredictable locations throughout the program. Certain Collector-produced tables, including the segment load table (SLT), are located at the beginning of the control bank. The C option on an I-BANK or D-BANK directive specifies that bank as the control bank. The control bank normally is defined as an initially-based D-BANK. If the C option is not present on any I-BANK or D-BANK directive, the Collector selects the first initially-based bank in the order of D-bank based on the Main PSR, D-bank based on the Utility PSR, I-bank based on the Main PSR, and I-bank based on the Utility PSR, to be the control bank.

2.2.5.5. Segmentation within Bank-Named Collections

A segmentation structure may be specified within each user-defined bank. If a bank has no segments, the bank is considered to be composed of one main segment. When segmentation does exist within banks, the same name must be used for the main segment of each bank. The segmentation structure within a bank may be entirely different from, or entirely the same as, the segmentation structure of another bank. A segment may be entirely contained within one bank, or it may span (be named and contained in) several banks. When a spanning segment is to be loaded, each portion of the segment is loaded into its respective bank. A bank-implied program, which has segments defined, may be thought of as a two-bank program (one I-bank and one D-bank) with each bank having the same segment structure, and each segment spanning the two banks.

Relocatable segments cannot span banks and must be included in their entirety in one bank.

2.2.5.6. Element Inclusion

2.2.5.6.1. Global Element Inclusion

All of the information on element inclusion and file searching for a bank-implied collection (see 2.2.3.2) is applicable to bank-named collections with a few important additions. When banks are not named in the map, any element to be included is split, with its odd location counters going to the I-bank (instruction area) and its even counters to the D-bank (data area). However, in a

bank-named collection, in addition to naming an element for inclusion, the user may also designate which location counters of the element are to be included within a bank. Normally, the selected location counters of an element are determined by the location counter set which is active at the time the element is named for inclusion (see 2.2.2.19). The selection can be overriden on an individual basis by using the $lcs field for the element named on the IN directive (see 2.2.2.1). Thus, the element name can appear on several IN directives, so long as different location counters are included each time the element is named.

All location counters of an element are included in a collection. Therefore, if an element has a location counter which was not specified for inclusion, that location counter is placed in the main segment of the control bank.

Example:

```
I-BANK      I1
$1,5
IN      ELTA
D-BANK      D1
IN      ELTA,ELTB
I-BANK      I2
$7,9,3
IN      ELTB($ODD),ELTA
$ALL
IN      ELTC,ELTD
```

The following shows which location counters of each element are placed in the various banks:

	I1	I2	D1
ELTA	1,5	7,9,3	EVEN,11
ELTB	--	ODD	EVEN
ELTC	--	ALL	--
ELTD	--	ALL	--

Note that the even counters of ELTA and ELTB are placed in D1 ($ODD assumed for I-BANKs, $EVEN for D-BANKs, unless otherwise specified). Also note that the odd counters for ELTB go to I2, overriding the active location counter set of $7,9,3. Location counter 11 in ELTA was placed in the control bank because no direction had been given as to its placement.

2.2.5.6.2. Local Element Inclusion

Local element inclusion is a feature to allow the use of multiple copies of elements within a given collection. The locality of these elements is determined by the specification of a local bank set list on the IN directive (see 2.2.2.1). The 'localness' of the elements named on such an IN directive is accomplished by allowing its undefined references to be satisfied and its external tags to satisfy external references only from elements contained within the set of banks defined in the local bank set. This set of banks consists of those banks named in the local bank list of the IN directive, plus the bank named in the preceding I-BANK or D-BANK directive.

The entry points of a local element satisfy only references made by other elements within the banks to which it is local. No other banks have access to these entry points. Within a given bank, elements and location counters can be included once and only once. If a version name is used with an element name and the element is included more than once, the same version name must be used for all inclusions of the element.

The purpose of this feature is to allow duplication of heavily–used routines without causing entry point conflicts and without forcing unduly repetitive switching of PSR bank bases to base very short pieces of heavily– used code.

Entry points can thus appear many times locally (satisfying references from within the local bank set), but can appear only once globally (satisfying references from all elements not named locally). Entry points named on the COR, SNAP, EQU, ENT, and DEF collector directives apply to the global specification of that entry point.

Any element desired for local inclusion must be named on the IN directive along with the local bank set list. No implicitly included element can be included locally.

2.2.5.7. Element Placement

Elements (or location counters of elements) which are named on IN directives, are placed within the bank which is named on the preceding I–BANK or D–BANK directive.

Although elements may be split by location counters between banks, the entire element; i.e., all its location counters, will be included somewhere in the absolute element. If not all of the location counters are specified on IN directives or on the $lcs directives, then the remaining location counters are placed in the main segment of the control bank (see 2.2.5.4).

Another method of element placement is defined for those elements which are included in the collection because their entry points satisfy external references from another element to be included in the collection. The placement of these implicitly included elements is dependent upon the placement of the referencing elements.

For bank–implied collections, the implicitly included elements are split with odd location counters to the I–bank and even location counters to the D–bank. For bank–named collections, the following rules apply:

1. In the case where there are at most two banks, based only on the Main PSR and no dynamic banks (see Volume 2–3.4.4.4.3), then the implicit element is split by odd and even location counters if there are two banks, or else included in its entirety in the one defined bank.

2. If the conditions of 1. are not satisfied and the implicitly included element is referenced only from within one bank; i.e., the referencing elements are contained in their entirety in this one bank, then the implicit element is split with the odd counters going to the referencing bank and the even counters going to the main segment of the control bank.

3. If the conditions of 2. are not satisfied and the implicitly included element is referenced from an element or elements which are not entirely contained in one bank, then the implicitly included element is placed in its entirety into the main segment of the control bank.

The user may wish to override the Collector's placement of implicitly included elements. This can be done by specifying on the LIB directive the bank or banks in which elements implicitly included from this file are to be placed (see 2.2.2.3).

2.2.5.8. Loading Program Segments

When a segmented program produced by a bank–named collection is called by an @XQT control statement, only the main segment of each static and initially based dynamic bank (see Volume 2–3.4.4.4.3) is loaded. The main segment of any other dynamic bank is automatically loaded upon execution of an LIJ or LDJ to that bank.

As in bank-implied collections, all subordinate segments are loaded by either:

- the direct method, or
- the indirect method.

2.2.5.8.1. Direct Method (L$OAD and LOAD$)

Segments within bank-named collections are directly loaded in the same manner as segments within a bank-implied collection (see 2.2.4.5.1). However, a consideration in bank-named collections is that each portion of that segment will be loaded into its respective bank. Thus, all banks which contain portions of that segment must be either static or currently based on the Main or Utility PSR at the time the segment is directly or indirectly loaded.

2.2.5.8.2. Indirect Method

Paragraph 2.2.4.5.2 describes the manner in which segments are indirectly loaded. The same considerations for directly loading segments in bank-named collections (see 2.2.5.8.1) are applicable when indirectly loading segments.

When referenced, any globally and locally defined entry points within the I-bank portions of an indirectly loaded segment will cause that segment to be automatically loaded (if it is not already loaded).

The reference to an entry point, which will cause the indirect load of a segment cannot be from the same element that contains that entry point even if that portion of the element making the reference is in another segment of the program.

2.2.5.8.3. Reloading the Main Segment in Bank-Named Programs

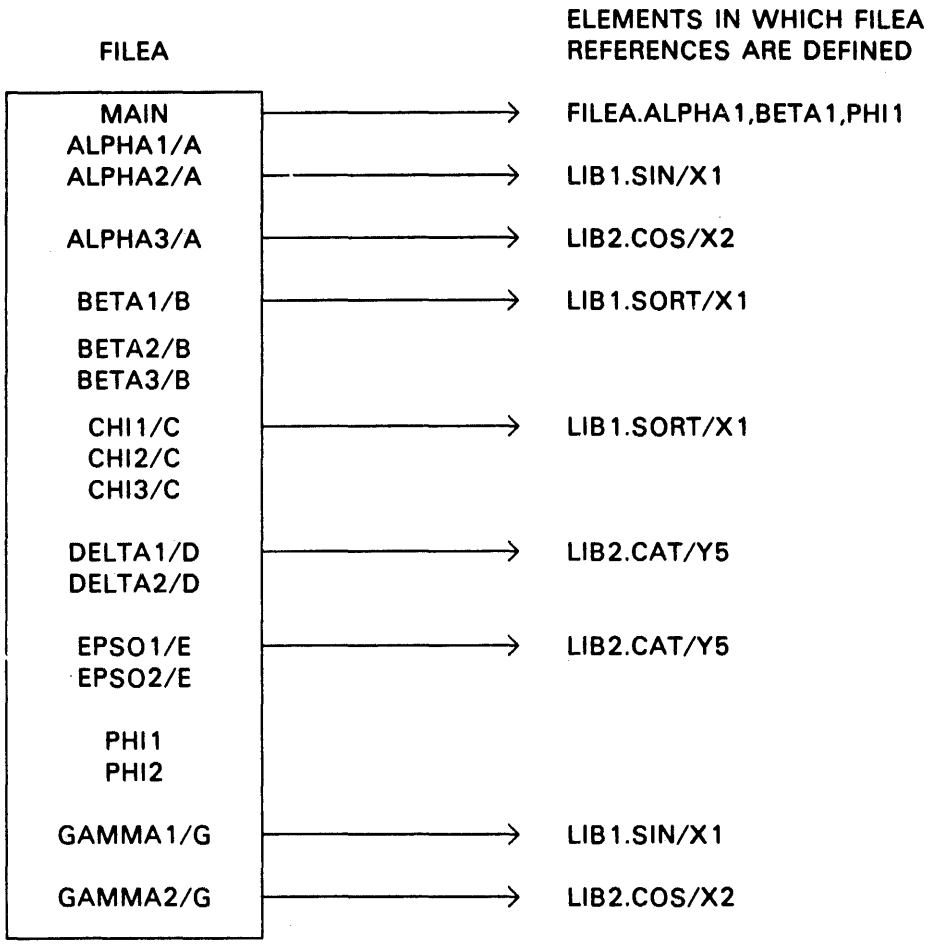
In both bank-implied and bank-named programs, the same calling sequence is used to reload the main segment (see 2.2.4.5.3).

However, the following apply to the main segment reload in bank-named programs:

- Main segment is reloaded for all static banks and any dynamic banks which were initially based.
- Any initially based dynamic banks must be based at the time of the main segment reload. If this is not true, the program will error terminate.
- Any currently based dynamic banks which were not initially based will not have their main segments reloaded.
- At the time of the reload, the current main storage requirements for all banks in which the main segment is reloaded cannot be less than the banks' initial main storage requirements.
- All banks which are based at the time of a main segment reload are still based after the reload has been performed.

2.2.6. Segmentation Example

The following is an example of a segmented program. The elements in the file FILEA and their required outside references are shown below:



The following coding is used to produce the segmented program:

```
1.  @PREP          LIB1.
2.  @PREP          LIB2.
3.  @MAP,IL        MAPSYM,MAPABS
4.  SEG            MAIN
5.  IN  FILEA.MAIN
6.  SEG            ALPHA*, (MAIN)
7.  IN  FILEA.ALPHA1/A,.ALPHA2/A,.ALPHA3/A
8.  SEG            BETA*, (ALPHA)
9.  IN  FILEA.BETA1/B,.BETA2/B,.BETA3/B
10. SEG            CHI*,BETA
11. IN  FILEA.CHI1/C,.CHI2/C
12. SEG            DELTA*, (BETA,CHI)
13. IN  FILEA.DELTA1/D,.DELTA2/D
14. SEG            EPSO*, DELTA
15. IN  FILEA.EPSO1/E,.EPSO2/E
16. SEG            GAMMA*, (MAIN)
17. IN  FILEA.GAMMA1/G,.GAMMA2/G
18. SEG            PHI*, (DELTA,GAMMA)
19. IN  FILEA.PHI1,.PHI2
20. LIB            LIB1,LIB2
21. END
```

1,2. Entry point tables are prepared for files LIB1 and LIB2.

3. Calls the Collector. The I option specifies that symbolic element MAPSYM is introduced from the runstream. The L option specifies that a complete listing is to be produced. The absolute output element is called MAPABS. Both MAPSYM and MAPABS are placed in TPF$.

4. Segment MAIN is the program's main segment.

5. Element MAIN is found in FILEA file.

6. Segment ALPHA is marked for indirect loading. The starting address of ALPHA follows the last address of segment MAIN.

7. Elements ALPHA1/A, ALPHA2/A, and ALPHA3/A are found in FILEA file and are included in the collection of ALPHA segment.

8. Segment BETA is marked for indirect loading. The starting address of BETA follows the last address of segment ALPHA.

9. Elements BETA1/B, BETA2/B, and BETA3/B are found in FILEA file and are included in the collection of BETA segment.

10. Segment CHI is marked for indirect loading. The starting address of CHI segment is the same as the starting address of segment BETA.

11. Elements CHI1/C and CHI2/C are in FILEA and are included in the collection of the CHI segment.

12. Segment DELTA is marked for indirect loading. The starting address of DELTA segment follows the last address of either segment BETA or segment CHI, whichever is longer.

13. Elements DELTA1/D and DELTA2/D are in FILEA file and are included in the collection of segment DELTA.

14. Segment EPSO is marked for indirect loading. The starting address of EPSO segment is the same as the starting address of segment DELTA.

15. Elements EPSO1/E and EPSO2/E are iN FILEA file and are included in the collection for segment EPSO.

16. GAMMA segment is marked for indirect loading. The starting address of GAMMA segment follows the last address of segment MAIN.

17. Elements GAMMA1/G and GAMMA2/G are in FILEA file and are included in

18. Segment PHI is marked for indirect loading. The starting address of PHI segment follows the last address of either segment DELTA or segment GAMMA, whichever is longer.

19. Elements PHI1 and PHI2 in FILEA file are included in the collection of PHI segment. the collection of segment GAMMA.

20. Files LIB1 and LIB2 are searched prior to searching the system library for the collection elements.

21. End of the Collector directives.

Figures 2-1 and 2-2 show the instruction and data areas of main storage for the preceding example.

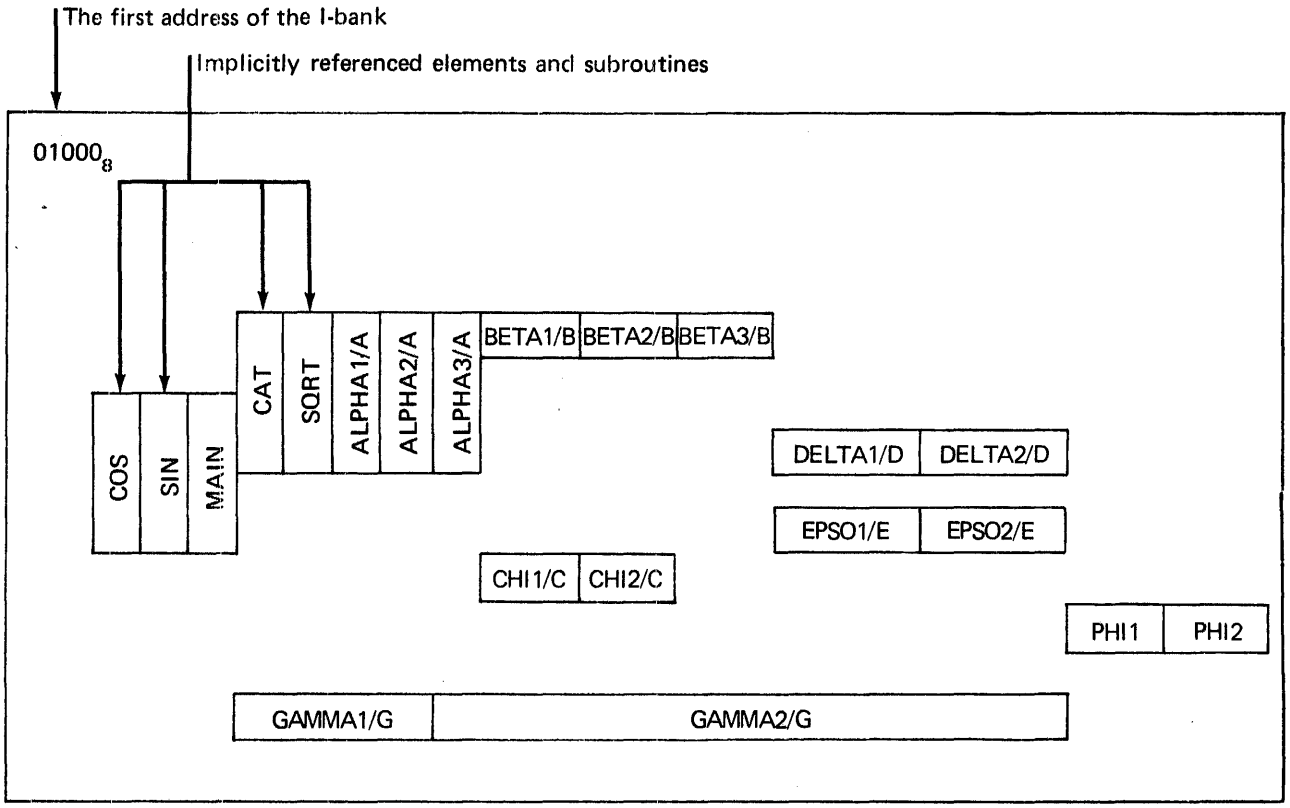


Figure 2-1. Instruction Area (I-Bank) Main Storage Map Segmented MAPABS

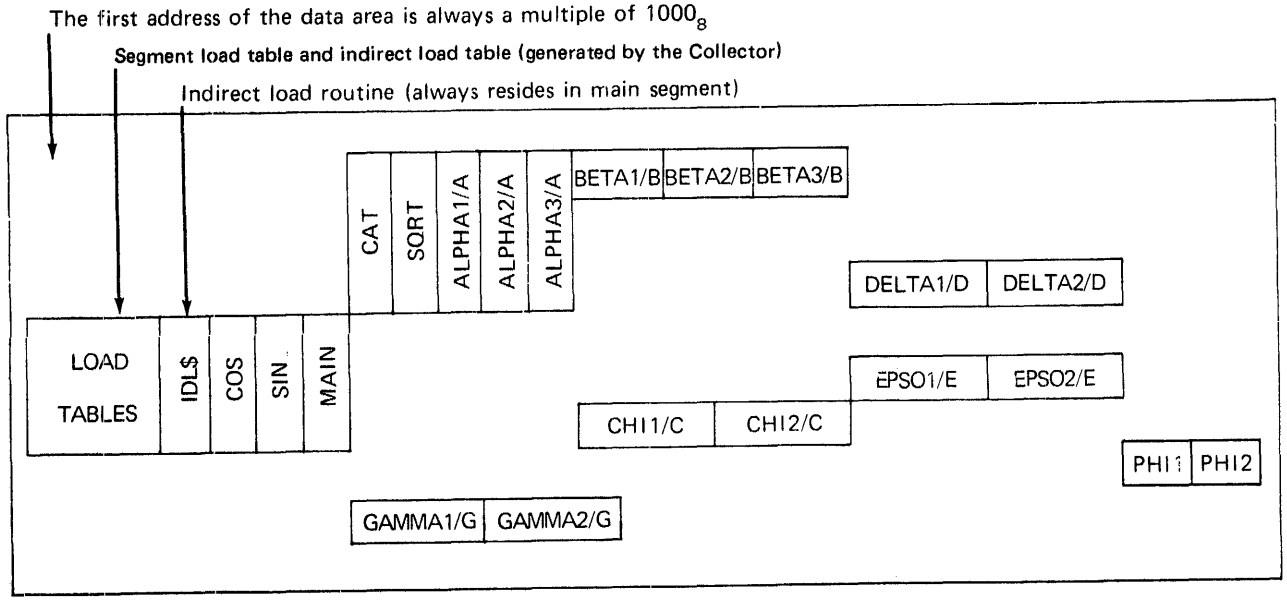
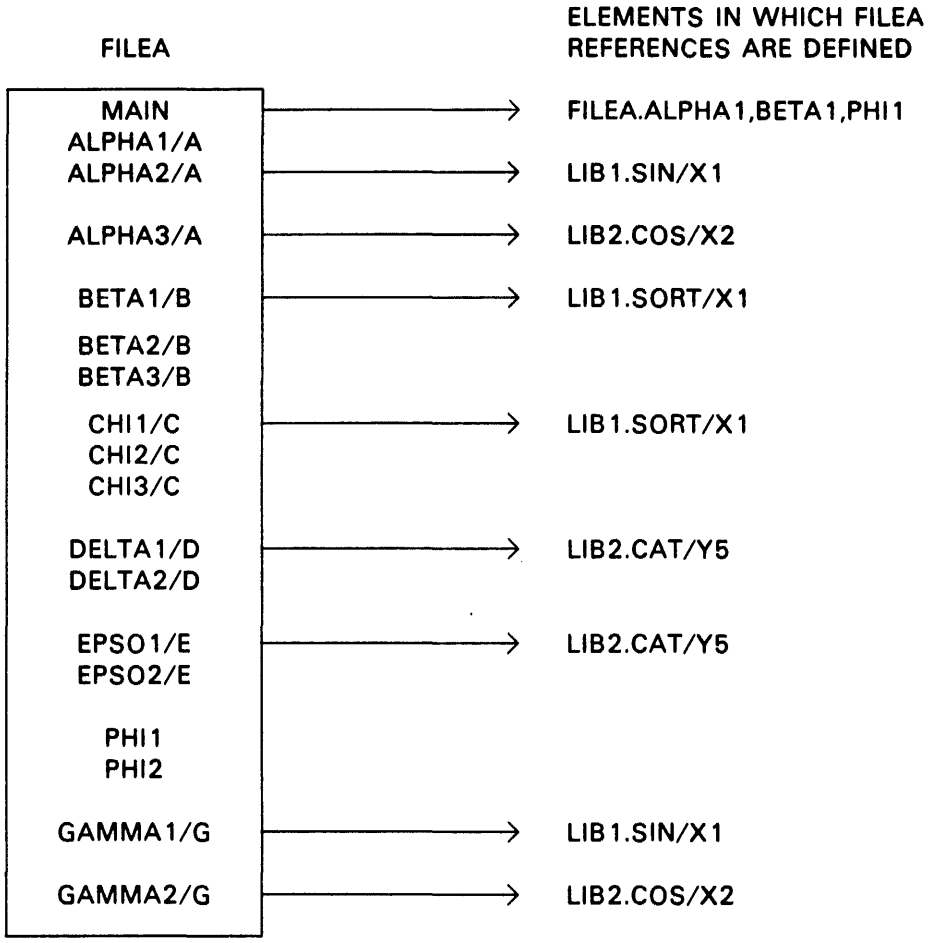


Figure 2-2. Data Area (D-Bank) Main Storage Map Segmented MAPABS

2.2.7. Bank-Named Segmentation Example

The following is an example of a bank-named segmented program. The elements in file FILEA and their required outside references are shown below:



The following coding is used to produce the bank-named segmented program:

```
1.  @PREP          LIB1.
2.  @PREP          LIB2.
3.  @MAP,IL        MAPSYM,MAPABS
4.  I-BANK,MD      BANK1
5.  SEG            MAIN
6.  IN             FILEA.MAIN
7.  SEG            ALPHA
8.  IN             FILEA.ALPHA1/A,.ALPHA2/A
9.  SEG            BETA*
10. IN             FILEA.BETA1/B
11. D-BANK,MC      BANK2
12. SEG            MAIN
13. IN             FILEA.MAIN
14. SEG            ALPHA
15. IN             FILEA.ALPHA1/A,.ALPHA2/A
16. SEG            BETA*
17. IN             FILEA.BETA1/B
18. IN             FILEA.BETA2/B($ALL)
19. I-BANK         BANK5,(BANK1)
20. $ALL
21. SEG            MAIN
22. IN             FILEA.CHI3/C
23. SEG            EPSO
24. IN             FILEA.EPSO1/E,.EPSO2/E
25. SEG            PHI
26. IN             FILEA.PHI1,.PHI2
27. I-BANK,U       BANK3,BANK5-02000
28. SEG            MAIN
29. IN             FILEA.BETA3/B
30. SEG            CHI
31. IN             FILEA.CHI1/C,.CHI2/C
32. SEG            DELTA,CHI
33. IN             FILEA.DELTA1/D,.DELTA2/D
34. D-BANK,U       BANK4
35. FORM           BANK3
36. D-BANK         BANK6,BANK4
37. $ALL
38. SEG            MAIN
39. IN             FILEA.GAMMA1/G
40. SEG            GAMMA
41. IN             FILEA.ALPHA3/A,.GAMMA2/G
```

1,2. Entry point tables are prepared for files LIB1 and LIB2.

3. Calls the Collector. The I option specifies that symbolic element MAPSYM is introduced from the runstream. The L option specifies that a complete listing is to be produced. The absolute output element is called MAPABS. Both MAPSYM and MAPABS are placed in TPF$.

4. I-bank BANK1 is a dynamic bank which is based on the Main PSR.

5. Segment MAIN is the main segment for BANK1.

6. The odd location counters from element MAIN in FILEA are to be included.

7. Segment ALPHA follows segment MAIN.

8. The odd location counters from elements ALPHA1/A and ALPHA2/A in FILEA are to be included.

9. Segment BETA follows segment ALPHA and is indirectly loaded.

10. The odd location counters from element BETA1/B in FILEA are to be included.

11. D-bank BANK2 is the control bank and is based on the main PSR.

12. Segment MAIN is the main segment in BANK2.

13. The even location counters from element MAIN in FILEA are to be included.

14. Segment ALPHA follows segment MAIN.

15. The even location counters from elements ALPHA1/A and ALPHA2/A in FILEA are to be included.

16. Segment BETA is indirectly loaded and follows segment ALPHA.

17. The even location counters from element BETA1/B in FILEA are to be included.

18. All location counters from element BETA2/B in FILEA are to be included.

19. I-bank BANK5 is a static bank and follows BANK1.

20. The $lcs specification is for the inclusion of all location counters of an element.

21. Segment MAIN is the main segment of BANK5.

22. All location counters from element CHI3/C in FILEA are to be included.

23. Segment EPSO follows segment MAIN.

24. All location counters from elements EPSO1/E and EPSO2/E in FILEA are to be included.

25. Segment PHI follows segment EPSO.

26. All location counters from elements PHI1 and PHI2 in FILEA are to be included.

27. I-bank BANK3 is a static bank based on the Utility PSR. Its starting address is equal to the starting address of BANK5 minus 02000.

28. Segment MAIN is the main segment of BANK3.

29. The odd location counters from element BETA3/B in FILEA are to be included.

30. Segment CHI follows segment MAIN.

31. The odd location counters from elements CHI1/C and CHI2/C in FILEA are to be included.

32. Segment DELTA starts at the same address as segment CHI.

33. The odd location counters from elements DELTA1/D and DELTA2/D in FILEA are to be included.

34. D-bank BANK4 is a static bank based on the Utility PSR. It follows D-bank BANK2.

35. The same segmentation structure and element inclusions for BANK3 are to be placed in BANK4, except that the even location counters of the elements are to be included rather than the odd location counters as in BANK3.

36. D-bank BANK6 is a static bank which starts at the same address as BANK4.

37. The $lcs specification is for the inclusion of all location counters for the following included elements.

38. Segment MAIN is the main segment of BANK6.

39. All location counters of element GAMMA1/G in FILEA are to be included.

40. Segment GAMMA follows segment MAIN.

41. All location counters from elements ALPHA3/A and GAMMA2/G in FILEA are to be included.

Figures 2-3 through 2-9 show the bank structure of the program, the segment structure within each bank, and the element inclusion within the segments and banks.

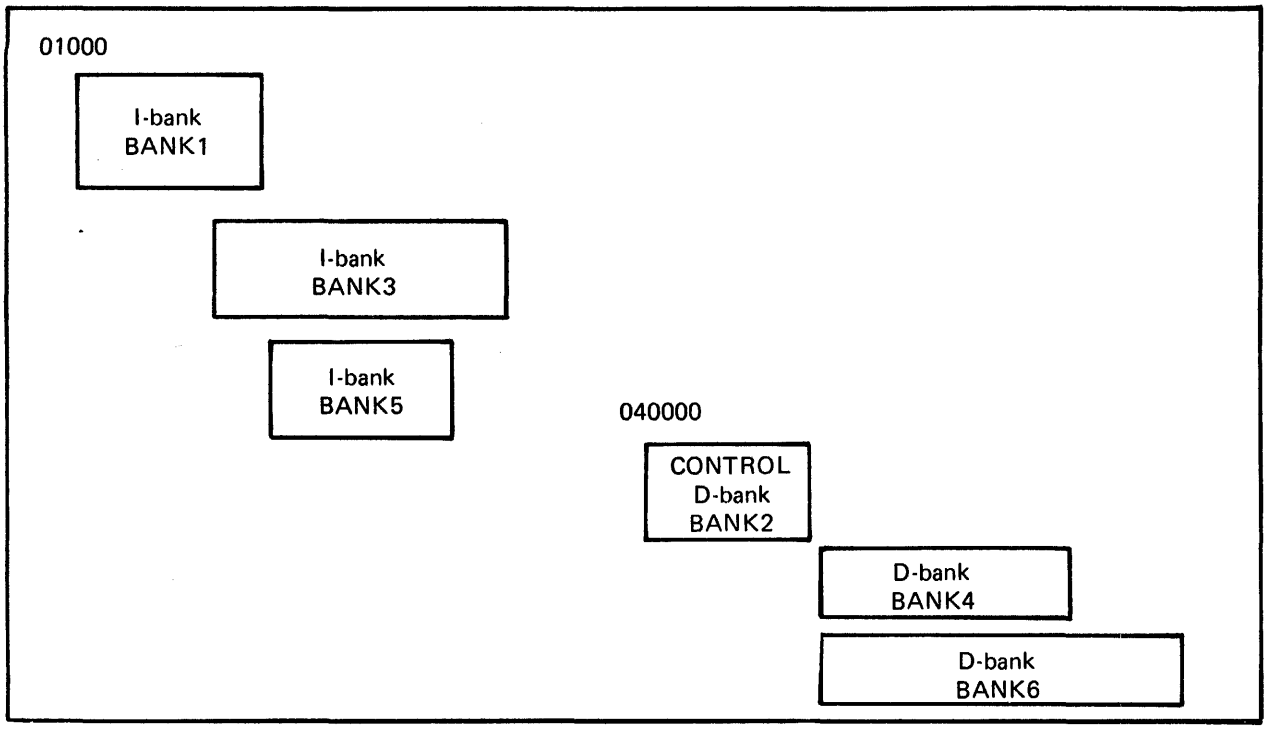


Figure 2-3. Bank Structure of Program and Segment Structure Within Each Bank

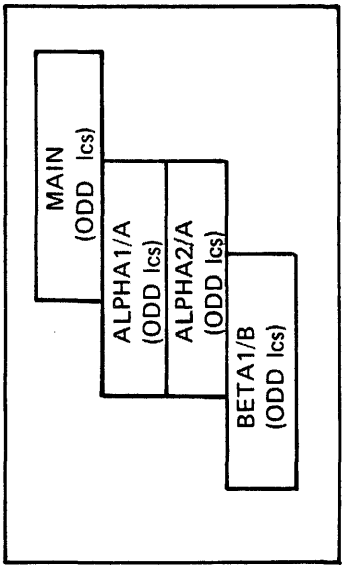


Figure 2-4. BANK1

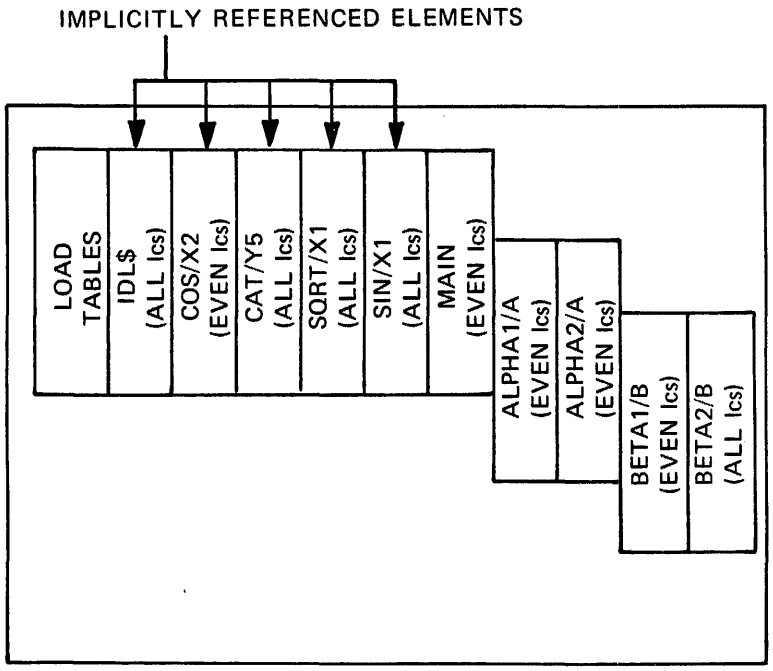


Figure 2-5. BANK2 (Control Bank)

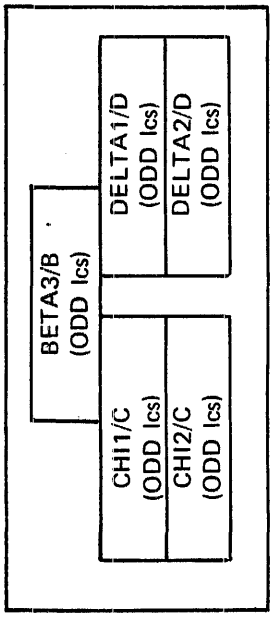


Figure 2-6. BANK3

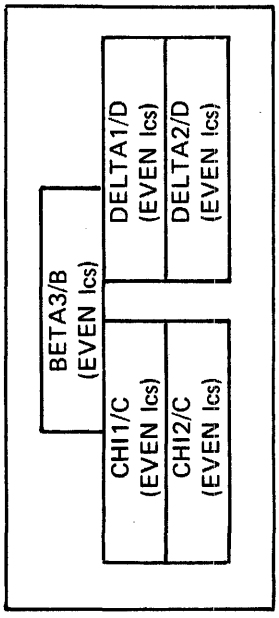


Figure 2-7. BANK4

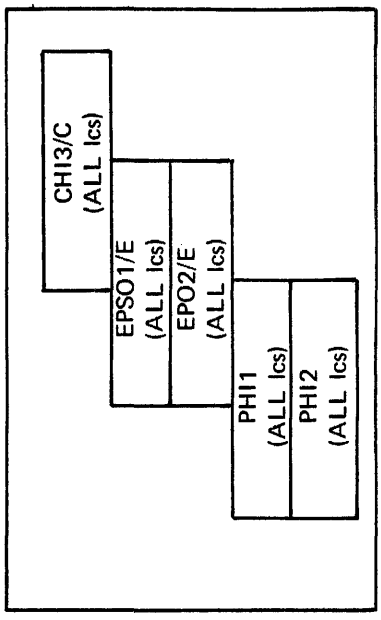


Figure 2-8. BANK5

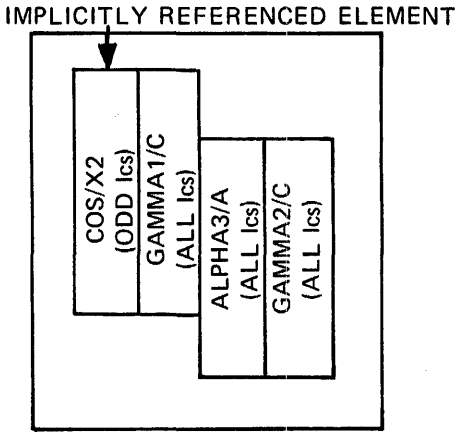


Figure 2-9. BANK6

2.2.8. Collector Generated Tables

Entry Point Table (ENTRY$):

Word		
0	*000000*	*nbr-of-entries*
1	*entry-point-name*	
2		
3	*value-of-entry-point-program-addr*	

The value of the entry point is the address of the reference vector entry of the entry point if in a segment designated for indirect loading.

The Entry Point Table includes only those entry points specified on the DEF statement.

Common Block Table (COMMN$):

Word		
0	*000000*	*nbr-of-entries*
1	*common-block-name (BLANK$COMMON for-blank-common)*	
2		
3	*length-of-common-block*	*addr-of-common-block*

When the DEF statement is present, the Common Block Table is included in the absolute program.

External Reference Table (XREF$):

Word		
0	*000000*	*nbr-of-entries*
1	*external-reference-name*	
2		
3	*ER ERR$*	

The external reference is assigned the address of the third word of its entry in the External Reference Table entry.

NOTE:

The first addresses of the Entry Point Table, Common Block Table, and External Reference Table are assigned, respectively, to the following external definitions (which may be directly referenced in a user program): ENTRY$, COMMN$, and XREF$. If no table exists, this value is zero.

Segment Load Table (SLT$)

Two formats of the Collector-produced Segment Load Table exist.

Type 1 format is produced for bank-implied collections.

Segment Load Table Format - Type 1

Word				
0	*A*	*type*	*0*	*forward link to active segment*
1	*last I-bank address*			*first I-bank address*
2	*last D-bank address*			*first D-bank address*
3	*0*		*sector address of first load control group*	

where:

A Bit 35 set if segment is not loaded.

Type	000	Main segment for type 1 SLT format
	010	Dynamic segment
	027	Relocatable segment
	024	Overlay segment

If the S2 value of word 0 of the first SLT entry (SLT$) is equal to zero, the table only contains four-word entries as formatted above.

If S2 of word 0 = 022 in the first SLT entry, the table format contains extension entries, in addition to four-word entries.

Type 2 format is produced for all bank-named collections.

Segment Load Table Format – Type 2

Word				
0	*A*	*type*	*0*	*forward link to active segment*
1	*last bank address*			*first bank address*
2	*BDI*		*0*	*extension index*
3	*0*		*sector address of first load control group*	

where:

A	Bit 35 set if segment is not loaded.	
TYPE	022	Main segment for type 2 SLT format
	011	Dynamic segment
	027	Relocatable segment
	024	Overlay segment
BDI	BDI value for bank into which segment part is loaded.	

When H2 of word 2 is nonzero, it contains a link to the next SLT extension for the segment.

NOTE:

The extension index points to the word immediately preceding the first word of the extension entry.

Word			
0	*last bank address*		*first bank address*
1	*BDI*	*0*	*extension index*

Each 4-word entry and its extensions are linked in order of ascending BDI value.

The SLT 4-word entry for the main segment never contains an extension index as no extension entries are built for the main segment.

For RSEG SLT entries, the format is the same in both the type 1 and type 2 segment load tables. The format is as follows:

Word				
0	*A*	*027*	*0*	*forward link to active segment*
1	*last rseg address*			*0*
2	*0*			*nbr of relocation words*
3	*0*		*sector address of first load control group*	

2.2.9. Collector Defined Tags

References to the following tags are satisfied by the Collector during collection:

ENTRY$ – First address of the Entry Point Table.

COMMN$ – First address of the Common Block Table.

XREF$ – First address of the External Reference Table.

SLT$ – First address of the Segment Load Table.

FRSTI$ – Lowest I-bank address assigned to the program.

FRSTD$ – Lowest D-bank address assigned to the program.

LASTI$ – Highest I-bank address assigned to the program, including dynamic segments.

LASTD$ – Highest D-bank address assigned to the program, including dynamic segments.

FIRST$ – Lowest address assigned to the bank in which the reference is made.

LAST$ – Highest address assigned to the bank in which the reference is made.

BDI$ – Bank descriptor index for the bank in which the reference is made.

BDIREF$ – Bank descriptor index for the bank in which the associated tag is defined.

BDICALL$ – As BDIREF$, unless the collection is bank-implied or the reference is in the same bank in which the tag is defined. In these cases, the value is zero. If the value of the associated tag is an absolute 36-bit number, then the top 18 bits are used.

IBJ$ – Used in the f-field of an instruction. LIJ operation code if the tag in the u-field of the instruction is an absolute value, or is defined in a bank other than that in which the reference is made, except for bank-implied collections. LMJ operation code if the collection is bank-implied, or the tag in the u-field is defined in the same bank as that in which the reference is made.

DBJ$ – Same as IBJ$ except that LDJ is generated in place of LIJ.

D$ATE – Satisfied with the date, in TDATE$ format, that th absolute element was created.

T$IME – Satisfied with the time, in TDATE$ format, that the absolute element was created.

D$ATE and T$IME provide 18-bit values for the date and time of the absolute element creation which may be edited using the EDIT$T or AEDIT$T packages to provide a means of verifying which version of a program is being executed.

2.2.9.1. Use of BDICALL$/IBJ$ Feature

BDICALL$ and IBJ$, together with the TYPE IBJLNK parameters, provide a powerful means of coding subroutines and their calls in such a way that the correct linkages and parameters may be generated by the Collector. This relieves the programmer of having to know when coding calls to subroutines, especially for library routines used in high level languages; whether the subroutine is to be collected in the same bank as the call, in a different bank in the program, or resides in a common bank. Also using TYPE IBJLNK, linkages may be set up to a common routine, where parameters to be passed are specified at collection time.

BDICALL$ and IBJ$ are used in the following format, where the instructions must be in the given order but not necessarily sequential. IBJ is assumed to be a proc that generates a word in instruction format with bits 26-35, the f-j fields relocated by the XREF IBJ$. X11 is used for convenience in the examples, however, any index register could be used.

```
LXI,U     X11,BDICALL$+TAG     . Load Parameter
IBJ       X11,TAG              . Call Subroutine
```

The following cases, used to determine how BDICALL$ and IBJ$ are satisfied, are defined.

- The calling sequence and TAG are both defined in the same bank, the collection is a bank implied collection, or the bank in which TAG is uefined in is specified with the 'L' option or is the control bank.

  In this case BDICALL$+TAG is satisfied as 0 and IBJ$ is satisfied as LMJ, such that the calling sequence is generated as if it were:

  ```
  LXI,U     X11,0
  LMJ       X11,TAG
  ```

- TAG is defined as an entry point, in a bank named collection, in a bank other than that in which the calling sequence is defined.

In this case BDICALL$+TAG is satisfied as the Bank Descriptor Index of the bank defining TAG – e.g., BDI1, and IBJ$ is satisfied as LIJ, such that the sequence is generated as if it were:

```
LXI,U    X11,BDI1          . LOAD BANK BDI
LIJ      X11,TAG           . ENTER NEW BANK
```

- TAG is defined as a 36-bit value in an element not specified on a TYPE IBJLNK statement.

  In this case the reference is assumed to be to a common bank, where bits 18–35 are the BDI and bits 0–17 are the entry point for the common bank, e.g., TAG was defined as:

```
TAG*     EQU  0400231002010      . BDIC,BNKENTRY
```

  BDICALL$ is satisfied as the BDI value from the 36-bit value. IBJ$ is satisfied as LIJ, and TAG is truncated to 18 bits. The sequence is generated as if it were:

```
LXI,U    X11,0400231       . BDIC
LIJ      X11,002010        . BNK ENTRY
```

- TAG is defined as an absolute value in an element which was specified on a TYPE IBJLNK statement in the collector source, e.g.,

```
TYPE          IBJLNK     SUBPARAM,SUBENTRY
```

  TAG is defined in SUBPARAM and SUBENTRY is the name of an entry point in the collection.

  In this case BDICALL$ is satisfied with the value of TAG, IBJ$ is satisfied as LMJ, and the address field relocated by TAG on the IBJ instruction is satisfied by SUB ENTRY. The sequence is generated as if it were:

```
LXI,U    X11,TAG           . LOAD PARAMETER

LMJ      X11,SUBENTRY      . CALL SUBROUTINE
```

  In this way many calls to SUBENTRY may be coded, specifying that different parameters are to be used on entry, but the parameters may be defined in a separate element, and may be changed without having to recompile the elements containing the subroutine calls.

When BDICALL$ and IBJ$ are satisfied in one of the first three ways, i.e., without the use of a TYPE IBJLNK specification, the called subroutine may easily determine the type of call and return appropriately. This is done by examining the value passed in the increment portion (H1) of X11, as follows:

```
TAG .
    S          X11,RETURN        . Save return address.
    •
    •                            . Process subroutine.
    •
    L          X11,RETURN        . Restore return address.
    TZ,H1      RETURN            . Call by LIJ or JUMP
    LIJ        X11,0,X11         . LIJ, LIJ back
    J          0,X11             . LMJ, JUMP back
```

2.3. PROGRAM EXECUTION

2.3.1. Initiating Execution (@XQT)

The @XQT statement (see Volume 2-3.4.4) is used to initiate the execution of an absolute element prepared by the Collector. The absolute program will be loaded into main storage.

See Volume 2-3.4.4 for a discussion on the following:

- Initial execution
- Initial execution status

  Initial PSR and storage limits

  1. Overlapped addresses
  2. Lowest bank address
  3. Initially-based common banks
  4. Common bank access

- Program data separation
- Bank referencing

  Visible banks
  Switching between banks
  Static versus dynamic banks
  Initial load
  PCT referencing

- Program termination

2.4. REENTRANT PROCESSOR EXECUTION

2.4.1. General

Reentrant processors are provided only to be compatible with earlier Executive systems. The functions provided by reentrant processors can better be provided by common or dynamic banks.

A reentrant processor (REP) is an executable reentrant routine referenced from a user's program by the LINK$ or RLINK$ Executive Requests. A reentrant processor consists of only I-bank addresses and resides as an absolute element in the system library (SYS$*LIB$) or a user file. A REP may be referenced many times during a user run without being reloaded and may access other banks of the calling program. For purposes of debugging, a reentrant processor may reside on mass storage in a user specified file. There are two types of reentrant processors:

- System standard reentrant processors listed in system generation
- User-specified reentrant processors listed by the RLIST$ Executive Request

See Volume 2-4.8.5 for further discussion of reentrant processor Executive Requests.

3. Debugging Aids

3.1. INTRODUCTION

The operating system provides a comprehensive set of diagnostic routines to aid in the checkout and debugging of user programs. The routines provided are:

- Postmortem Dump (PMD) Processor
- Dynamic Dump Routines
- Program Trace Routine (SNOOPY)
- Flow Analysis Program (FLAP)

Another diagnostic capability, the production of snapshot dumps, is provided by the SNAP$ request (see Volume 2–4.10.3) and the Collector's SNAP directive (see 2.2.2.10). Snapshot dump output, like program trace (SNOOPY) output, is placed in the user's print file at the point at which the request was made. The output of the dynamic dump routines is handled by the PMD processor; the output of the PMD processor is listed after the print output generated by the user's program. SNOOPY and FLAP are described in Volume 4.

A diagnostic file (DIAG$), which is used for recording diagnostic information for the PMD processor, is automatically assigned to each run by the system. This file is divided into two functional parts: a dynamic dump portion and PMD portion. The dynamic dump portion always starts at the beginning of the diagnostic file and it is followed by the PMD portion.

The dynamic dumps consists of dumps of:

- main storage,
- control registers,
- magnetic tape files, and
- mass storage files.

The postmortem dump consists of the final contents of a program's main storage area at termination.

3.2. POSTMORTEM DUMP PROCESSOR (PMD)

The Postmortem Dump Processor (PMD) is called by the @PMD control statement. At program termination the final contents of the program's main storage areas are written into the diagnostic file by the system. The information can then be edited and printed by the PMD processor. Postmortem dumps may be taken of

- overlay segments,
- elements,
- banks, or
- any portion of the terminated program as long as those segments, elements, banks, and/or portions are active when the program terminates.

See Volume 2–3.4.4.4.5 for the description of which program banks are considered active. Within a program bank, a portion of the terminated program is considered active if it is either the main segment or an overlay segment which is loaded as described in 2.2.4.5.1. or 2.2.4.5.2.

3.2.1. @PMD Control Statement

Purpose:

Calls the PMD processor to dump all or specified portions of a program that was in main storage at program termination. Any number of PMD control statements may follow the execution of a program so that different parts of a program's storage area may be dumped in different formats, if desired.

Format:

```
@PMD,options    operand fields
```

Parameters:

options

general	Applies to all types of PMD control statements. The presence of A, I, or D options indicate a format 3 PMD control statement. The absence of operand fields indicates a format 4 PMD control statement.
special	Applies to format 3 or 4 PMD control statements. PMD control statements without A, I, or D options and having operand fields are either format 1 or 2. Format 1 PMD control statements are used to dump all or part of a specific location counter of a specific element. Format 2 PMD control statements are used to dump areas starting at specified externally defined entry points.

If no parameters are specified in the operand fields, all elements residing in main storage at program termination are dumped in accordance with the specified options.

Format 1:

```
@label:PMD,options    eltname/bankname,address/lc,length,format
```

Format 2:

@label:PMD,options epname/bankname,length,format

Format 3:

@label:PMD,options part-1,part-2,...,part-n

where part-n may be of the following forms:

eltname
segment
bankname
eltname/segname
eltname/bankname
segment/bankname
eltname/segname/bankname

Format 4:

@PMD,options

NOTE:

The presence of an A, I, or D options indicates a format 3 PMD.

Parameters:

options — The general options in Table 3-1 apply to any format PMD control statement. All of the special options in Table 3-2 apply to format 3 PMD control statements. On format 4 PMD control statements the following special options may be used: I, D, L, F, G, N, O, Q, and S. If more than one of the format options (F, G, N, O, Q, and S) are used on format 3 or 4 PMD control statements, then each location counter of each element dumped is repeated, once for each type of format requested. If no format options are specified on format 3 or 4 PMD control statements, then octal format is used.

eltname — Specifies the element to be dumped. The names of labeled common blocks or BLANK$COMMON may also be used. Common blocks can be considered to have one location counter (location counter zero).

epname — Specifies the entry point name from which the dump is to start. This name must be externally defined and referenced by another element in the program. If any externally defined name is to be referenced in a format 2 PMD, then the user program must be MAPped to produce extended diagnostic tables using the TYPE EXTDIAG directive (see 2.2.2.13).

segname — Specifies the segment to be dumped.

bankname — Specifies the bank to be dumped. If used as subfield containing other names, it defines which bank of the multibank program to consider for dumping.

part — Specifies the element, segment or bank to be dumped. For multibank programs with localized elements or segments which span banks, subfields may be used to define the specific portion of the element or segment to be dumped. The subfield names must maintain the same relative sequence (i.e., bank name last, segment name preceding bank name, etc.) although each of the subfields is optional and any type of name can appear in any subfield.

address — The address, relative to the beginning of the location counter (lc), at which the dump should begin. This field applies only to format 1. If this field is omitted, an address of zero is assumed.

lc — Specifies which location counter of the element to be dumped. If this parameter is omitted, a location counter of zero is assumed.

length — Specifies the number of words to be dumped. If this parameter is omitted, the word length in the location counter being dumped is assumed.

format — The single letter which references one of the standard editing formats (see 3.3.1.8.1) to be used. Optionally, a FORTRAN format expression enclosed in parentheses may be defined by the user in this field to be used as the editing format.

Any format statement acceptable to FORTRAN V for output editing may be used, with the exception of one using the R editing code. See 1100 Series FORTRAN V, UP-4060 (current version). In addition, N and S may be used as editing codes in format statements for mnemonic and octal instruction formats, respectively. When N is used as an editing code in a FORTRAN format statement, at lease 26 spaces must be provided for each word edited. When S is used as an editing code, at least 21 spaces must be provided for each word edited. Note that the D standard format and user-supplied formats are not applicable to changed word dumps. If the parameter is omitted, an octal dump is produced.

The address, location counter and length may be specified in octal or decimal number notation. (Numbers with a leading zero are assumed to be octal.)

NOTE:

Since element names need not be unique to the program, care must be taken in requesting PMD processing. The element name may be the same as the entry point name, segment name or bank name. The order of search within the PMD porcessor is as follows:

1. *Bank Name*
2. *Segment Name*
3. *Element Name*
4. *Entry Point Name*

Therefore, if unique element names are not used, care must be taken in requesting dumps to ensure that the proper information is obtained. In order to prevent possible conflicts, it is recommended that unique element names be used whenever possible.

Table 3-1. @PMD Control Statement, General Options

Option Character	Description
C	Dumps the words which were changed during the execution or loading of the allocated program area of main storage specified in the @PMD control statement.
E	Processes @PMD control statement only if the previous routine terminates in error.
M	Print only diagnostic dumps that have not been printed. This applies to demand runs only.
P	Causes an octal dump of the PCT blocks used by the run to be printed preceding the dump of the program. Also the segment load tables and any other collector generated table, if any, are dumped in octal.
B	Dumps information about a program's banks. Information dumped: bank's name, BDI, base address, bank type, and storage preference.
T	Formats the output of PMD so that no print line is longer than 80 characters. This option is intended for use with output devices such as the UNISCOPE 100, but use is not limited to demand runs. When the T option is used on the first PMD control statement following the execution of a program, it also controls the editing of any Dynamic Dumps produced.
W	Debug only - Turns on snap dumps in PMD.
Y	Debug only - Turns on snap dumps in PMD.

Table 3-2. @PMD Control Statement, Special Options

Option Character	Description
A	Produces a dump of the specified main storage area of each named element or segment or bank.
D	Produces a dump of the D-bank portion of each named element, segment or bank. If no names are specified, then all of the D-bank portion of the program is considered for dumping.
I	Produces a dump of the I-bank portion of each named element, segment or bank. If no names are specified, then all of the I-bank portions of the program is considered for dumping.
L	Dumps the active library elements. When the A, I or D options are used, the L option is necessary if any of the elements named in the specification fields are system library elements. @PMD,L dumps all active elements including those from the system library.
X	Used in conjunction with the A, I or D options. Dumps all active elements except those named in the control statement and those belonging to the segments named in the control statement.
F	Produces a dump in Fieldata alphanumeric format (see Table 3-3).
G	Produces a dump in G format (see Table 3-3).
N	Produces a dump in mnemonic instruction format (see Table 3-3).

Table 3-2. @PMD Control Statement, Special Options (continued)

Option Character	Description
O	Produces a dump in octal format. This option is used only when any of the F, G, N, Q, and S options are used. Produces an octal dump in addition to the other formats requested.
Q	Produces a dump in ASCII alphanumeric format (see Table 3-3).
S	Produces a dump in ASCII octal instruction format (see Table 3-3).

Description:

For the A, I, D, and X options, the names of labeled common blocks or BLANK$COMMON may be used as element names.

See Volume 2-3.4.1 for the effect of the @RUN control statement on postmorten dumps.

If no information was saved by the system when the previous execution terminated, no dumps are possible. This condition is caused by an N option on the @RUN control statement. A PMD is not possible for a program if the Z option was specified to the Collector on the @MAP control statement when that program was mapped. If no dump is available, a message is produced.

In demand or batch mode, all control statements are allowed between the normal termination of an @XQT or processor call and the @PMD request for the @XQT or processor call. Additionally, all control statements will be allowed to intervene between @PMD requests. If another @XQT or processor call is encountered, it will be honored and DIAG$ will be rewritten when the @XQT or processor call terminates. The above also applies to all types of termination in demand mode.

When in batch mode, if an @XQT or processor call terminates in error, the @PMD control statement will be honored only if it follows the @XQT or processor call. Only data statements, @EOF control statements and the conditional control statements (@SETC, @JUMP, @TEST and @ADD) may intervene. @PMD control statements are not honored for batch runs that terminate in abort mode.

The standard SYS$*LIB$ version of the PMD processor is never written to DIAG$.

Example:

```
@XQT        PROGX
data
  •
  •
  •
  •
data
@TEST       TE/6/S3
@JUMP       3
@SETC       6/S4
@PMD,A      ELEMENT1, ELEMENT2
@XQT        PROGY
```

If PROGX terminates before processing all of the data statements that follow the first @XQT control statement, and S3 of the condition word has a value of 6,S4 of the condition word is set to 6, and the @PMD and @XQT PROGY control statements are processed.

Any @PMD control statement following the execution of a program from SYS$*LIB$ (that is, @FOR, @COB, @MAP, and so forth) is honored only if the Y option appears on the @RUN statement.

However, if the W option was specified on the @RUN control statement when in batch mode, any program from SYS$*LIB$ that error terminates will be automatically written to DIAG$ and the PMD processor will automatically be loaded to print the erroring program. The PMD processor is called with P and L options specified.

Examples:

```
1.  @PMD
2.  @PMD,EAXL       PETER,BOB
3.  @PMD,D          FLYBY
4.  @PMD,ECD        REPORT
5.  @PMD            ALPHA,100/3,56,A
6.  @PMD            TOP/CAT,10,A
7.  @PMD,I
8.  @PMD,ILNO
9.  @PMD,DLOFG
10. @PMD,ILNO       BETA/SEGA,GAMMA/SEGB/BANKA,SEGC
11. @PMD,AOFG       BLANK$COMMON,DELTA
```

1. An octal dump of all active (allocated in main storage and loaded) segments of a user's program results. Active elements in the program which were from SYS$*RLIB$ are not included in the dump.

2. An octal dump of all active elements, except PETER, BOB, and active system library element results. The dump occurs only if the previous routine terminated because of an error.

3. Results in an octal dump of the D-bank of segment FLYBY (if active).

4. Causes an octal dump of changed words in the D-bank portion of element REPORT (if active). The dump occurs only if the previous program terminated because of an error.

5. The result of this @PMD control statement is a 56-word dump of location counter 3 of element ALPHA in the standard alphanumeric editing format. The dump begins with relative address 100 under location counter 3.

6. The dump begins at external entry point TOP in bank CAT. Ten words are dumped in the standard alphanumeric editing format.

7. Dump the I-bank portion of all active elements except those included in the program from the System Library.

8. Dump the I-bank portion of all active elements including those within the program from the System Library. Each location counter of each element is dumped twice: once in octal format, and once in mnemonic instruction format.

9. Dump the D-bank portion of all active elements. Each location counter of each element is dumped three times: once in Fieldata alphanumeric format, once in octal format, and once in G format.

10. Dump the I-bank portion of element BETA in segment SEGA, element GAMMA in SEGB in bank BANKA, and the I-bank portion of all elements in segment SEGC including those from the System Library. Each location counter of each element dumped is repeated twice: once in octal format, and once in mnemonic instruction format.

11. Dump BLANK COMMON and all portions of element DELTA, BLANK COMMON and each location counter of element DELTA are dumped three times: once in octal format, once in Fieldata alphanumeric format, and once in G format.

3.3. DYNAMIC DUMPS

The dynamic dumps are discussed from the viewpoint of the SPERRY UNIVAC 1100 Series Assembler user. There is no inherent restriction on the employment of this facility with any other processor. All that is needed is that the proper information be written to the diagnostic file. Library routines are provided to assist in the process. The use of the dynamic dump facility by a high level language processor fails outside the scope of this manual.

Dynamic diagnostic requests are generated by procedure–calls from within the user program. These procedure calls collect the dynamic diagnostic library subroutines in to the user object program. The requested dynamic diagnostic information is written into the diagnostic file by the library subroutines while the object program is being executed. When called on during program execution, these subroutines preserve the complete program environment and perform the requested dynamic diagnostic request. If the user's program has multiple activities, only one activity at a time may execute dynamic dump calls since the dynamic dump routines are non–reentrant.

The amount of information which can be written into the dynamic diagnostic portion of diagnostic file can be set dynamically through the use of the X$SIZE procedure (see 3.3.3.5). If this procedure is not used, the length will automatically be limited by the value specified in the system's generation.

When the dynamic diagnostic portion of diagnostic file is filled, a message is supplied indicating that no more dynamic diagnostics can be transferred to the diagnostic file. All subsequent dynamic diagnostic requests for this program are ignored. After program termination in batch mode, the dump information is retrieved from the diagnostic file, edited and printed. When in demand mode, a message will be displayed informing the user that diagnostic dumps are available and the name of the program that produced the dumps. The user may then call the PMD processor with an M option to retrieve, edit, and print the dynamic dumps only; or without the M option to retrieve, edit, and print both the diagnostic dumps and a dump of the program.

There are 19 different library subroutines associated with the dynamic diagnostic procedures. These routines can be divided into three functional classifications:

1. Dump Procedures: X$MESG, X$CW, X$CORE, X$DUMP, X$TAPE, X$DRUM, X$FILE, and X$CREG which are used to record data in the diagnostic file.

2. Conditional Control Procedures: XIF, XAND, X$OR, and X$TALY which are used to determine when a given dump or series of dumps should occur.

3. Specification Procedures: X$FRMT, X$BUFR, X$MARK, X$BACK, X$SIZE, X$ON, and X$OFF which are used to specify the condition switch and other global diagnostic parameters.

3.3.1. Dump Calling Procedures

The procedures available for obtaining dynamic dumps are:

XCORE$ (see 3.3.1.1)
XDUMP$ (see 3.3.1.2)
XCW$ (see 3.3.1.3)
XTAPE$ (see 3.3.1.4)
XDRUM$ (see 3.3.1.5)
X$FILE (see 3.3.1.6)
XCREG$ (see 3.3.1.7)

All dynamic dump procedures are executed only if the switch XSTAT$ (see 3.3.3.2) is on.

The dynamic dump procedures save and restore all control registers as well as the carry and overflow conditions. The dynamic dump routines contained in SYS$*RLIBS are not reentrant. Therefore, only one activity of a program should reference these procedures at a time.

3.3.1.1. Main Storage Dump (XCORE$)

Purpose:

Produces a dump of the specified main storage area.

Format:

```
        SLJ     XCORE$
N$      FORM    4,14,18
        N$      index-reg,word-count,starting-addr
        +       'format',0
```

This linkage may be generated by the procedure call:

```
X$CORE  starting-addr,word-count,'format',index-reg
```

Parameters:

starting-addr	Specifies the main storage starting location of the dump.
word-count	Specifies the number of locations to be dumped (037777 maximum).
'format'	Specifies a single letter, enclosed in quotes, which references either a standard (see 3.3.1.8.1) or a user-defined (see 3.3.1.8.2) editing format. If omitted, an octal dump is produced.
index-reg	Specifies the index register used to modify the address specified by the starting-addr parameter. This parameter, which may be omitted or left zero, can be set to values from 1 to 15 to specify an index register from X1 through A3. The value in the index register is added to the starting-addr value to get the actual dump starting address.

Description:

The main storage dump printout is preceded by the heading: **CORE DUMP**

Examples:

```
1.  X$CORE      TABLEX,100,'0',X5
2.  X$CORE      TABLEY,150
3.  SLJ         XCORE$
    N$          FORM      4,14,18
    N$          X10,250,WW3X
    +           'A',0
```

1. The main storage dump begins at address TABLEX as modified by index register X5. The dump is 100 words in length and is presented in standard octal format.

2. The main storage dump begins at address TABLEY, has a word length of 150, and is presented in standard octal format.

3. The main storage dump begins at address WW3X as modified by index register X10. The dump is 250 words in length and is presented in alphanumeric format.

3.3.1.2. Control Register and Main Storage Dump (XDUMP$)

Purpose:

Produces a dump of the program environment, A, X, and R registers, and main storage.

Format:

```
        SLJ     XDUMP$
N$      FORM    4,14,18
        N$      index-reg,word-count,starting-addr
        +       'format',register-code
```

This linkage may be generated by the procedure call:

```
X$DUMP  starting-addr,word-count,'format','AXR',index-reg
```

Parameters:

starting-addr	Specifies the main storage starting location. If omitted, a starting location of zero is assumed.
word-count	Specifies the number of locations to be dumped (037777 maximum). If omitted, a length of zero is assumed and no main storage dump is produced.
'format'	Specifies a single letter, enclosed in quotes, which references either a standard (see 3.3.31.8.1) or a user-defined (see 3.3.1.8.2) editing format. If omitted, an octal dump is produced.
'AXR'	Specifies, enclosed in quotes, one or more letters representing the A, X, and R registers. The contents of these registers are printed in octal.

index-reg — Specifies the index register used to modify the address specified by the starting-addr parameter. This parameter, which may be omitted or left zero, can be set to values from 1 to 15 to specify an index register from X1 through A3. The value in the index register is added to the starting-addr value to get the actual dump starting address.

register-code — Register codes for XDUMP$ are:

No registers	0_8
R only	200401_8
A only	200202_8
R and A	400603_8
X only	200104_8
X and R	400505_8
X and A	400306_8
A,X, and R	600707_8

Description:

The printout resulting from XDUMP$ is preceded by the heading: **DUMP**

The following additional information is provided following the **DUMP** heading:

- element name
- location counter
- relative program address
- hardware fault indicators

Example:

```
1.  X$DUMP      TABLEY,200,'I','XA',X10
2.  X$DUMP      TABLEZ,500,'A','R'
3.  X$DUMP      ,,,'R'
4.  SLJ         XDUMP$
    N$          FORM      4,14,18
    N$          X9,500,BSS6
    +           '0',    0200104
```

1. The main storage dump begins at address TABLEY as modified by index register X10. The dump is 200 words in length. The contents of all X and A control registers are also dumped. The dump is presented in the standard integer format.

2. The main storage dump begins at address TABLEZ and has a length of 500 words. The contents of all R control registers (except R0) are also dumped. The dump is presented in the standard alphanumeric format.

3. The contents of all R control registers (except R0) are dumped in octal format.

4. The main storage dump begins at address BSS6 as modified by index register X9. The dump is 500 words in length. The contents of the X registers are also dumped. The dump is presented in octal format.

3.3.1.3. Changed Word Dump (XCW$)

Purpose:

Produces a changed word dump of specific locations within main storage. On the first X$CW call referencing a given main storage area, a complete dump of that area is produced. On subsequent X$CW calls specifying the same area, only those words which were changed since the last X$CW procedure call specifing that area are dumped showing the previous contents and the current contents. The number of separate areas that may be dumped is restricted to five. The starting-addr and length determine the uniqueness of one area from the next.

Format:

```
SLJ     XCW$
+       word-length,starting-addr
+       'format',0
```

This linkage may be generated by the procedure call:

```
X$CW   starting-addr,word-length,'format'
```

Parameters:

starting-addr	Specifies the main storage starting location of the dump.
word-length	Specifies the number of locations to be dumped (037777 maximum).
'format'	Specifies a single letter, enclosed in quotes, which references one of the following standard editing formats: A, E, F, G, I, N, O, Q, and S (see 3.3.1.8.1). Standard format D and user-defined formats cannot be specified. If omitted, an octal dump is produced.

Description:

The number of calls on X$CW is not limited, but only five separate areas may be dumped.

Changed word dumps, whether or not any words were changed are preceded by the following heading plus the appropriate changed-word status word message:

```
**CHANGED WORD CORE DUMP**
```

Examples:

```
1.  X$CW        INSERT,10,'I'
2.  X$CW        REWORD,50
3.  SLJ         XCW$
    +           750, HTR5
    +           'F', 0
4.  X$CW        INSERT,10,'I'
```

1. The changed word dump begins at address INSERT. The dump is 10 words in length and is presented in standard integer format. Since this is the first call specifying an area starting at INSERT, all of the area is dumped.

2. The changed word dump begins at address REWORD. The dump is 50 words in length and is presented in standard octal format. Since this is the call specifying an area starting at REWORD, all of the area is dumped.

3. The changed word dump begins at address HTR5. The dump is 750 words in length and is presented in fixed-point decimal format.

4. The changed word dump begins at address INSERT. Any words in the 10-word area starting at address INSERT which were changed since dump number 1 occured are printed showing the previous and current contents.

3.3.1.4. Tape Block Dump (XTAPE$)

Purpose:

Dumps the block of magnetic tape data located just prior to the current tape position by making temporary use of a previously defined buffer initialized by the X$BUFR procedure (see 3.3.3.1). The magnetic tape is moved backward one block, the block is read, and the number of words specified in the X$BUFR procedure are dumped.

Format:

```
SLJ     XTAPE$
+       word-count,buffer-addr
+       'format',I/O-pktaddr
```

This linkage may be generated by the procedure call:

```
X$TAPE  I/O-pktaddr,'format'
```

Parameters:

I/O-pktaddr	Specifies the address of the I/O request packet (see Volume 2-6.2) for the device handler. This parameter may be the address of a file control table (FCT) as is used by block buffering and other routines, since the first six words of an FCT is an I/O packet.
'format'	Specifies a single letter, enclosed in quotes, which references either a standard (see 3.3.1.8.1) or user-defined (see 3.3.1.8.2) editing format. If omitted, an octal dump is produced.

Description:

Interblock gaps separate the blocks that are recorded on magnetic tape each time an I/O write of any size word count is done. These interblock gaps, which serve as block separators, are not to be confused with end-of-file (EOF) marks, which are a special kind of block surrounded by interblock gaps. The X$TAPE procedure causes a move backward to the preceding interrecord gap, then a read of everything which follows (could be one word or tens of thousands of words) into the buffer initialized by an X$BUFR procedure (see 3.3.3.1) until the next interrecord gap is encountered. When the buffer is filled, the remaining words are lost.

The X$TAPE procedure is useful for dumping a block that was just read or written. No dump occurs if the magnetic tape is positioned at the load point (beginning-of-tape marker).

No magnetic tape dump occurs if a main storage buffer is not reserved and initialized for the X$TAPE procedure.

The same buffer area can be used for both X$DRUM (see 3.3.1.5) and X$TAPE procedure calls.

The word count and buffer address are returned by the X$TAPE procedure to the first parameter word.

The tape drum printout is preceded by the heading:

```
*TAPE DUMP

**FILE filename
```

Example:

```
1.  X$BUFR      ALPHA,150
2.  X$TAPE      FILEA,'0'
3.  X$BUFR      BUF8,800
4.  SLJ         XTAPE$
    +           800,  BUF8
    +           '0',  NP16
```

1. The block of data prior to the present magnetic tape position is read into the main storage location ALPHA (previously initialized by the X$BUFR procedure call) and is printed in standard octal format. FILEA specifies the I/O packet address. If the block is longer than 150 words, the first 150 words are dumped.

2. The block of data prior to the present magnetic tape position is read into the main storage location BUF8 and is printed in standard octal format. NP16 specifies the I/O packet address.

3.3.1.5. Mass Storage Dump (XDRUM$)

Purpose:

Dumps portions of FASTRAND drum-formatted mass storage by making temporary use of a previouly defined buffer initialized by the X$BUFR procedure (see 3.3.3.1). Portions of mass storage to be dumped and read into the buffer, then the contents of the buffer is written into the diagnostic file.

Format:

```
SLJ     XDRUM$
+       word-count,location-addr
+       'format',I/O-pktaddr
```

This linkage may be generated by the procedure call:

```
X$DRUM  I/O-pktaddr,location-addr,word-count,format
```

Parameters:

I/O-pktaddr — Specifies the address of the I/O packet containing the internal filename (see Volume 2-6.2).

location-addr — Specifies the address of a word which contains the relative starting sector address or a word address of the file to be dumped. (In some cases, this address may be I/O-pktaddr+5, which contains a sector address or a word address.) The manner in which the file was assigned determines whether the address specified is a word address or a sector address (see Volume 2-3.7.1.1 and 2-3.7.1.3).

word-count — Specifies the number of locations to be dumped.

'format' — Specifies a single letter, enclosed in quotes, which references either a standard (see 3.3.1.8.1) or a user-defined (see 3.3.1.8.2) editing format. If omitted, an octal dump is produced.

Description:

The mass storage dump printout is preceded by the heading:

DRUM DUMP FILE filename AT RELATIVE SECTOR sector-number

Use of the X$DRUM procedure requires a main storage buffer into which the mass storage dump can be read. The mass storage area to be dumped is read into the buffer. When it is filled, the contents of the buffer is written into the diagnostic file. For FASTRAND drum-formatted files, it is recommended that the buffer be some multiple of 28, the length of a FASTRAND drum mass storage sector. While a portion of mass storage that is larger than the size of the buffer may be dumped, greater efficiency results by providing a buffer that is sufficiently large to hold all the mass storage to be dumped at one time.

If a main storage buffer is not reserved and initialized for the X$DRUM procedure, no mass storage dump occurs.

The same buffer area can be used for both X$DRUM and X$TAPE (see 3.3.1.5 and 3.3.1.4) procedure calls.

Example:

```
1.  X$BUFR      DUMPB,112
    X$DRUM      FILED,DRDUMP,112,'A'
2.  X$BUFR      AREA1,450
    SLJ         XDRUM$
    +           450, LWA1
    +           'D', ADDR1
```

1. Beginning at the relative address value specified by DRDUMP, 112 words of data from mass storage are read into the buffer starting at main storage location DUMPB that was initialized by the X$BUFR procedure call. The data is edited in standard alphanumeric format. FILED specifies the I/O packet address.

2. Beginning at the relative address value specified by LWA1, 450 words of data from mass storage are read into the buffer starting at main storage location AREA1. The data is edited in double-precision, floating-point format. ADDR1 specifies the I/O packet address.

3.3.1.6. File Dump (X$FILE)

Purpose:

Provides for dumps in connection with the item handler package. Dynamic dumps of items can be taken whenever an item is read from or written into a particular file.

Format:

```
X$FILE  fct,option,'format'
```

Parameters:

fct — Specifies the address of the file control table.

option — The available options are:

'ON' – Causes subsequent items read from or written into the file to be dumped.

'OFF' – Terminates dumping of items from the file.

'format' — Specifies a single letter, enclosed in quotes, which references either a standard (see 3.3.1.8.1) or a user–defined (see 3.3.1.8.2) editing format. This parameter can be specified only when the ON option is specified. If omited, an octal dump is produced.

Description:

This procedure cannot be used for an item that spans multiple blocks.

Examples:

```
1.  X$FILE      BETA,'ON','O'
2.  X$FILE      BETA,'OFF'
```

1. The file whose file control table is located at BETA is conditioned to record in the diagnostic file all subsequent activity at the item level. That is, every time a request is made to an item, the item to which the item handler points is recorded in the diagnostic file and is printed in standard octal format.

2. The file whose file control block is located at BETA is conditioned not to dump any subsequent activity at the item level.

3.3.1.7. Control Register (User Set) Dump (XCREG$)

Purpose:

Dumps specified user control registers. (The A, X, and R registers and the unassigned registers at addresses 034 and 035).

Format:

```
SLJ     XCREG$
+       register–count,starting–reg
+       'format',0
```

This linkage may be generated by the procedure call:

```
X$CREG  starting–reg,reg–count,'format'
```

Parameters:

starting-reg	Specifies the address of the first control register to be dumped.
reg-count	Specifies the number of control registers to be dumped.
'format'	Specifies a single letter, enclosed in quotes, which references either a standard (see 3.3.1.8.1) or a user-defined (see 3.3.1.8.2) editing format. If omitted, an octal dump is produced.

Description:

The register dump printout is preceded by the heading: **CREG DUMP**

Examples:

```
1.  X$CREG      X1,12,'O'
2.  X$CREG      X11,10,'A'
3.  X$CREG      A14,10,'O'
```

1. Registers X1 through X11 and A0 are dumped into the diagnostic file to be edited and printed in standard octal format.

2. Control registers X11 and A0 through A8 are dumped into the diagnostic file to be edited and printed as a string of 60 alphanumeric (Fieldata) characters.

3. Control registers A14 and A15, the unassigned registers 034 and 035, and R0 through R5 are dumped into the diagnostic file to be edited and printed in standard octal format.

3.3.1.8. Editing Formats for Dynamic Dumps

Each procedure for calling dynamic dumps specifies an editing format for printing the dump. Standard editing formats (see 3.3.1.8.1) are available to the user for this purpose. If, however, the user desires to define the editing format, X$FRMT procedures must be used (see 3.3.1.8.2).

3.3.1.8.1. Standard Editing Formats for Dumps

A number of standard editing formats are available to the user when specifying dump procedures. These formats provide the majority of printing formats desired. Table 3-3 lists the standard formats, which are specified by a single letter enclosed in quotation marks in the dump procedure calls (see 3.3.1.1 through 3.3.1.7). Figure 3-1 is an example of printouts of integer and octal dumps in standard editing format.

a. Integer Format Dump

INSTRUCTION:

X$DUMP B1,32,'I'

PRINTOUT:

```
**DUMP**
  CALLING ELEMENT NAME$            $( 00) RELATIVE LOCATION OF CALL    000011
  PANEL       CARRY OFF    OVERFLOW OFF

  REGISTERS

  DUMP OF ELEMENT NAME$              $( 00)   AT MAP ADDRESS   040620     CREATED ON: 20 JUL 75 AT 13:53:29
000035    040620              3          6          11          20          37          70         135         264
000045    040630            521       1034        2059        4108        8205       16398       32783       65552
000055    040640         131089     262162      524307     1048596     2097173     4194326     8388631    16777240
000065    040650       33554457   67108890   134217755   268435484   536870941  1073741854  2147483679  4294967328
```

b. Octal Format Dump

INSTRUCTION:

X$DUMP B1,32,'O'

PRINTOUT:

```
**DUMP**
  CALLING ELEMENT NAME$            $( 00) RELATIVE LOCATION OF CALL    000011
  PANEL       CARRY OFF    OVERFLOW OFF

  REGISTERS

  DUMP OF ELEMENT NAME$              $( 00)   AT MAP ADDRESS,  040620     CREATED ON: 20 JUL 75 AT 13:53:29
000035    040620    000000000003 000000000006 000000000013 000000000024 000000000045 000000000106 000000000207 000000000410
000045    040630    000000001011 000000002012 000000004013 000000010014 000000020015 000000040016 000000100017 000002000020
000055    040640    000000400021 000001000022 000002000023 000004000024 000010000025 000020000026 000040000027 000100000030
000065    040650    000200000031 000400000032 001000000033 002000000034 004000000034 010000000036 020000000037 040000000040
```

Figure 3-1. Standard Editing Format for Integer and Octal Dumps, Sample Printout

Table 3-3. Standard Editing Formats for Dump Printouts

Format Parameter	Definition	Number of Items Per Line		Number of Print Positions Per Item	Number of Decimal Places
		T Option	NO T Option		
A	Alphanumeric editing of Fieldata data	8	16	6	-
D	Floating decimal editing of double precision floating point data	2	4	26	18
E	Floating decimal editing of single precision floating point data	4	8	14	8
F	Fixed point decimal editing of single precision floating point data	4	8	14	8
G	Fixed point or floating decimal editing of single precision floating point data	4	8	14	Variable
I	Decimal integer editing of 36-bit signed computer words	4	8	14	-
N	Mnemonic editing of 1100 Series instruction words	2	4	27	-
Q	Alphanumeric editing of ASCII data	8	16	4	-
S	Octal editing of 1100 Series Instruction - 6 fields per word	2	4	20	-

3.3.1.8.2. User-Defined Editing Formats (XFRMT$)

Purpose:

Specifies a nonstandard editing format for use by the diagnostic dump procedure calls as an alternative to the standard editing formats described in 3.3.1.8.1, or redefines the standard editing formats. New format labels (such as 'U', 'V', or 'W' for example) may be specified, or existing standard format labels may be redefined.

Format:

```
SLJ     XFRMT$
+       format-specification-word-length,'format-label'
'(format - specification)'
```

This linkage may be generated by the procedure call:

```
X$FRMT  format-specification-word-length,'format-label'
'(format - specification)'
```

Parameters:

format-specification-word-length — Specifies the number of words comprising the format specification.

'format-label' — Specifies a single letter enclosed in quotation marks. If one of the following standard editing formats: A, D, E, F, G, I, N, Q or S (see 3.3.1.8.1) is referenced, this action is used to redefine the standard editing formats. To specify a user-defined editing format, any letter (enclosed in quotes) except A, D, E, G, I, N, Q, or S may be used.

'(format-specification)' — Specifies a string of alphanumeric characters which represent an encoding of the format to be applied to the information printed. The string of alphanumeric characters may not contain intervening blanks. The first nonblank character of the string must be a left parenthesis (preceded by a quotation mark); the last nonblank character must be a right parenthesis (followed by a quotation mark).

The format of the string of characters that comprise this parameter is specified exactly as in FORTRAN V FORMAT statements. For example, specifying '(10F3.3)' indicates that the dump information printed on one line consists of 10 words of fixed-point decimal data and that each word is eight characters long with the decimal point at the left of the third least significant character.

Description:

Any standard or user-specified editing format may be redefined; the most recent definition prevails.

Multiple line formats are allowable.

Except for the 'R' conversion code, any format that can be given in a FORTRAN V FORMAT statement can be specified. See SPERRY UNIVAC 1100 Series FORTRAN V, UP-4060 (current version).

In addition, N and S may be used as editing codes in format statements for mnemonic and octal instruction formats, respectively. When N is used as an editing code in a FORTRAN format statement, at least 26 spaces must be provided for each word edited. When S is used an an editing code, at least 21 spaces must be provided for each word edited. Note that the D standard format and user-supplied formats are not applicable to changed word dumps.

Also, the 'S' and 'N' formats are available which edit each word in SPERRY UNIVAC 1100 Series mnemonic instruction format.

Examples:

```
1.  X$FRMT          1,'O'
    '(6O14)'
2.  X$FRMT          2,'F'
    '(10F8.3)'
3.  SLJ             XFRMT$
  - +               1, 'A'
    '(12A4)'
```

1. The standard octal editing format is redefined to print six octal words per line instead of eight. The appropriate data is written into the diagnostic file so that the redefined format is effective when the diagnostic editor processes the recorded dynamic data.

2. The standard fixed decimal format is redefined to print 10 fixed decimal words instead of eight. The number of characters per word is changed to eight instead of 14, and the number of decimal places is three instead of eight.

3. The standard alphanumeric editing format is redefined to 12 words per line instead of 16 and four characters per word instead of six.

3.3.2. Conditional Control Procedures

The dynamic dumps can be controlled by an internal conditional dump switch. When the switch is turned off by a conditional control procedure, a dynamic dump procedure which follows is ignored. Note that all dump procedures are executed except when preceding conditional dump procedures cause them to be overridden. A number of miscellaneous control procedures are available to the user in addition to the conditional control procedures.

The available conditional control procedures are:

X$IF (see 3.3.2.1)
X$OR (see 3.3.2.2)
X$AND (see 3.3.2.3)
X$TALY (see 3.3.2.4)

3.3.2.1. Logical IF Control of Dumps (X$IF)

Purpose:

Turns on or off the conditional dump switch depending on the value of the relational expression. Only a dynamic dump call, which immediately follows an XIF, XAND, or an X$OR call is affected by the setting of the conditional dump switch. Such a dump request is not executed if the conditional dump switch is off.

Format:

X$IF addr[,index-reg] [,j-desg] 'rel' addr[,index-reg] [j,-desig]

Parameters:

addr — Specifies a main storage location or a control register; indirect addressing and literals are allowed.

index–reg — Specifies an index register (X1 through X11, A0 through A3); index register incrementation is not allowed.

j–desig — Specifies any desired partial word.

'rel' — Specifies the relation between the parameters specified before and after 'rel'. Allowable codes for 'rel' are as follows:

Code	Meaning
'EQ'	Equal to
'GE'	Greater than or equal
'GT'	Greater than
'LE'	Less than or equal
'LT'	Less than
'NE'	Not equal

If the relation between the tested parameters is true, the conditional dump switch is turned on; if the relation is false, the conditional dump switch is turned off.

Examples:

```
1.  X$IF        ALPHA 'EQ' TAG
2.  X$IF      ALPHA,X1,H1 'LT' TAG,X1,H1
3.  X$IF        ALPHA,,H1 'NE' TAG,,H1
```

1. If the contents of ALPHA equals ('EQ') the contents of TAG, the conditional dump switch is turned on. If the contents of ALPHA does not equal the contents of TAG, the conditional dump switch is turned off.

2. If the contents of H1 of ALPHA, as modified by index register X1, is less than ('LT') the contents of the H1 of TAG, as modified by index register X1, the dump switch is turned on. If the modified contents of ALPHA is equal to or greater than the modified contents of TAG, the dump switch is turned off.

3. If the relationship of the contents of the H1 portions of ALPHA and TAG is not equal ('NE'), the dump switch is turned on; if the contents of the H1 portions of ALPHA and TAG are equal, the dump switch is turned off.

3.3.2.2. Logical OR Control of Dumps (X$OR)

Purpose:

Turns on the conditional dump switch if it is not already on and the current value of the relational expression is true. If the switch is initially on, it will remain on even if the relational expression is false.

Format:

```
X$OR  addr[,index-reg] [,j-desig]  'rel' addr[,index-reg] [,j-desig]
```

Parameters:

Same as X$IF procedure (see 3.3.2.1)

Examples:

```
1.  X$OR          ALPHA  'EQ'  TAG
2.  X$OR          ALPHA,X5,H2  'GT'  TAG,,H2
```

1. In this example, the conditional dump switch is set when the contents of ALPHA equals ('EQ') the contents of TAG.

2. The conditions for setting the conditional dump switch on are similar to those described in example 1. The condition being tested is greater than ('GT'); to turn the switch on, H2 of ALPHA as modified by index register X5 must be greater than H2 of TAG.

3.3.2.3. Logical AND Control of Dumps (X$AND)

Purpose:

Causes the conditional dump switch to remain on if, and only if, it is already on, and the current value of the relational expression is true.

Format:

```
X$AND  addr[,index-reg] [,j-desig] 'rel' addr[,index-reg] [,j-desig]
```

Parameters:

Same as X$IF procedure (see 3.3.2.1).

Examples:

```
1.  X$AND         ALPHA  'EQ'  TAG
2.  X$AND         ALPHA,,T1  'LE'  TAG,X10,T1
```

1. The conditional dump switch remains on if it is already on and the contents of ALPHA equals the contents of TAG.

2. The conditions for the conditional dump switch remaining on are similar to those described in example 1; the difference is that to remain on, T1 of ALPHA must be less than or equal to ('LE') T1 of TAG as modified by index register X10.

3.3.2.4. Controlling the Conditional Dump Switch (X$TALY)

Purpose:

Allows a dynamic dump procedure that is embedded in a loop to be executed only when conditions specified by the user are met. The conditional dump switch remains on when these conditions are met (if it is already on), and is turned off when they are not met.

Format:

```
X$TALY  start,until,every
```

Parameters:

start	Specifies the initial or starting value of the loop.
until	Specifies the maximum number of times the loop is to be executed.
every	Specifies a value which indicates the number of times the loop is to be executed before the conditional dump switch is turned on. For example, if the user specifies a value of 100, the conditional dump switch remains on every 100th time through the loop; all subsequent dynamic dump procedures in the loop are executed.

Description:

The X$TALY procedure is used to set the conditional dump switch by testing a counter. The counter is set to the value of the start parameter the first time the X$TALY procedure is executed. Thereafter, each time the procedure is entered for execution, the counter is incremented by one following all tests by the procedure. The tests performed on the counter (represented by the symbol Z) are:

if start $\leq$ (Z) $<$ until; and

if $\left\lfloor \frac{(Z) - \text{start}}{\text{every}} \right\rfloor$ yields a zero remainder

If the conditional dump switch is already on, it remains on if the tests are successful. If any of the tests fail, the switch is turned off.

The user should ensure that the conditional dump switch is on when an X$TALY procedure is entered; otherwise, the counter is not incremented and control is returned to the user program.

Example:

```
X$TALY          0,4000,100
```

In this example, 0 indicates the start of the loop, which is to be executed 4000 times. Every 100th time through the loop, up to 3999, the conditional dump switch remains on, provided it is on prior to execution of the X$TALY procedure. All other times the dump switch is turned off. Thus, the user obtains any specified dumps each 100th time through the loop.

3.3.3. Specification Procedures

A number of procedures in addition to conditional control procedures (see 3.3.2) are available to allow user control of dumps. These procedures are:

X$BUFR	(see 3.3.3.1)
X$ON and X$OFF	(see 3.3.3.2)
X$MARK and X$BACK	(see 3.3.3.3)
X$MESG	(see 3.3.3.4)
X$SIZE	(see 3.3.3.5)

3.3.3.1. Initializing Buffer (XBUFR$)

Purpose:

Initializes an area of main storage for use as a buffer by the X$TAPE or X$DRUM procedures (see 3.3.1.4 and 3.3.1.5, respectively).

Format:

```
SLJ     XBUFR$
+       word-count,starting-addr
```

This linkage may be generated by the procedure call:

```
X$BUFR  starting-addr,word-count
```

Parmeters:

starting-addr	Specifies the starting main storage address of the buffer.
word-count	Specifies the number of locations in the buffer to be initialized.

Description:

X$BUFR does not reserve a buffer area in main storage; it only initialzes the area. The buffer must be defined and initialized prior to executing any X$TAPE or X$DRUM procedure.

For dumps of FASTRAND drum-formatted file, it is recommended that the buffer be some multiple of 28, the length of a FASTRAND formatted mass storage sector.

Example:

```
X$BUFR      TDUMP,56
```

TDUMP is the main storage buffer area, 56 word in length, which is initialized for use by an X$TAPE or X$DRUM procedure.

3.3.3.2. Allowing and Ignoring Dump Procedure Calls (X$ON and X$OFF)

Purpose:

Allows overall control of the execution of dynamic dump procedure calls.

Format 1:

```
S       A0,XSTAT$
TNZ     XSTAT$
SN,H2   A0,XSTAT$
```

This linkage may be generated by the procedure call:

```
X$ON
```

Format 2:

```
SZ  XSTAT$
```

This linkage may be generated by the procedure call:

```
X$OFF
```

Description:

The word XSTAT$, which is in the XCOMN$ data area, is initially set to nonzero. If it should become desirable for all dynamic dump subroutines to return control immediately without processing any dumps, XSTAT$ may be cleared to zero by either the X$OFF procedure or the SZ instruction. To return XSTAT$ to its original nonzero status, the X$ON procedure or any equivalent instructions may be used.

The X$OFF procedure turns off the conditional dump switch until an X$ON procedure is executed. When the X$OFF procedure is executed, the setting of the conditional dump switch cannot be changed until the X$ON procedure is encountered. Thus the X$OFF procedure causes dynamic dump and conditional procedure call to be ignored, and the X$ON procedure allows the calls to be executed.

Care must be taken if dynamic dump procedures are used in programs consisting of independent activities and in I/O completion routines.

A series of dynamic dump procedure calls will not be interrupted by one of the other subprograms.

Examples:

```
1.  X$OFF
2.  X$IF        ALPHA 'EQ' TAG
3.  X$CORE      ALPHA,200,'E'
4.  X$ON
5.  X$CORE      TAG,150,'I'
```

The X$OFF procedure indicates that all diagnostic system dump procedures which follow, except the X$ON procedure, are to be ignored; therefore, the X$IF and X$CORE procedures (line 3) are not executed. The X$ON procedure indicates that all subsequent diagnostic system dump procedures are allowed; therefore, the X$CORE procedure (line 5) which follows is executed.

3.3.3.3. Saving and Deleting Dynamic Dumps (XMARK$ and XBACK$)

Purpose:

Marks the points in program execution between which dynamic dumps are saved and then deleted at the user's discretion. The X$MARK and X$BACK procedures permit a user program under

checkout to include dynamic dump procedures which the user may want to execute only when a routine does not terminate normally.

Format 1:

```
SLJ  XMARK$
```

This instruction may be generated by the procedure call:

```
X$MARK
```

Format 2:

```
SLJ  XBACK$
```

This instruction may be generated by the procedure call:

```
X$BACK
```

Description:

The $MARK and X$BACK procedures behave much as left and right parentheses surrounding portions of a program which are to be dumped only if termination occurs between them.

X$MARK and X$BACK pairs may be nested to a depth of five. The total number of occurrences of X$MARK and X$BACK is unrestricted.

Examples:

```
X$MARK
X$CORE          ALPHA,200,'A'
X$BACK
```

X$MARK saves the current location where the next write is to be made in the diagnostic file (by X$CORE). X$BACK resets the current location pointer to the value saved by the most recent X$MARK reference. The result is that all intervening dump information is overwritten by the next dump that is taken; that is, the data recorded by X$CORE is deleted.

3.3.3.4. Placing a Message in the Dump (XMESG$)

Purpose:

Permits the user to place any desirable message into the dynamic dump.

Format:

```
SLJ       XMESG$
+         word-length-of-msg,'A'
'diagnostic-msg'
```

This linkage may be generated by the procedure call:

```
X$MESG      word-length-of-msg
'diagnostic-msg'
```

Parameters:

word-length-of-msg	Specifies a number equal to the number of computer words in the message (one computer word hold six characters).
'diagnostic-msg'	Any string of Fieldata alphanumeric characters enclosed in quotes and printed exactly as assembled.
'A'	Generated by the procedure call. It is of no significance to the user, but it must be coded when the instruction form of the format is used.

Description:

The X$MESG procedure produces a line on the output listing of up to 120 alphanumeric characters. The printed line immediately follows the procedure reference.

The X$MESG procedure is executed only when the conditional dump switch is on.

Example:

```
X$MESG      5
'BEGIN TEST OF DIAGNOSTICS'
```

The five-word message is printed on the dynamic dump output.

3.3.3.5. Changing Length of Dump File (X$SIZE)

Purpose:

Changes the length of the area of the diagnostic file reserved for dynamic dumps.

Format:

```
SLJ     XSIZE$
+       length
```

This linkage may be generated by the procedure call:

```
X$SIZE  length
```

Parameters:

length	Specifies the length (in sectors) of the diagnostic file to be reserved for dynamic dumps.

Description:

Using this procedure, a user program can dynamically expand or contract the length of the dynamic dump portion of the diagnostic file. If this is not used, a system standard value is assumed for the length of the dynamc dump portion of the file. If this procedure is used, it should be used before executing dynamic dumps to ensure enough space for those dumps taken.

Example:

```
X$SIZE          2000
```

The length of the dynamic dump portion of the diagnostic file is changed to 2000 sectors.

3.3.4. Examples of Dynamic Dumping

The following example indicates the effect of conditional control procedures upon dump procedures. Note that if dump procedures are interspersed with conditional control procedures, they are effective only if the conditional dump switch is on at the time they are entered. Dump procedures have no effect on the setting of the conditional dump switch.

Assume that a program contains the variables X, Y, and Z (which have values 78, 80, and 88, respectively), and the constants A, B, and C (which have values of Fieldata characters A, 180, and 40_8, respectively). Also assume that the procedures in the following example are executed sequentially, and that they are the first group of procedures encountered.

Example:

	Coding	Conditional Dump Switch	Dump Taken
	.		
1.	$(1)		
	X$MESG 5	ON	
	'BEGIN TEST OF DIAGNOSTIC'	—	
	X$IF X 'EQ' A	OFF	
2.	X$MESG 4	OFF	
	'TEST DATA GROUP A'	—	
	X$BUFR DUMPB, 150 . INITIALIZE BUFFER	—	
	X$IF X 'EQ' A	OFF	
	X$OR X 'LT' Z	ON	
3.	X$CORE TABLEX, 100,'0'	ON	YES
4.	X$DUMP TABLEY,200,'I','XA'	ON	YES
5.	X$TAPE FILEA,'0'	ON	YES
	X$IF Y 'GT' B	OFF	
6.	X$TAPE FILEB,'0'	OFF	NO
	X$BUFR ALPHA,200 . INITIALIZE BUFFER	—	
	X$OR Y 'NE' Z	ON	
7.	X$TAPE FILEC,'0'	ON	YES
	X$BUFR ALPHA,112 . INITIALIZE BUFFER	—	
8.	X$DRUM FILED,DRDUMP,112,'A'	ON	
9.	X$FILE BETA,'ON'	ON	
10.	X$FILE BETA,'OFF'	ON	
11.	X$CREG 1,12,'0'	ON	YES
	$(2) .		
	DRDUMP + 0 . VALUE SET DYNAMICALLY BY USER		
	ALPHA RES 200		
	BETA (FILE CONTROL TABLE)		
	TABLEY RES 200		
	TABLEX RES 100		
	DUMPB RES 150 . BUFFER FOR DUMPING FROM TAPE AND DRUM		
	FILEA (EXEC I/O PACKET)		
	FILEB (EXEC I/O PACKET)		
	FILEC (EXEC I/O PACKET)		
	FILED (EXEC I/O PACKET)		

Explanation:

1. The message BEGIN TEST OF DIAGNOSTIC is recorded in the diagnostic file because the conditional dump switch is on.

2. The message TEST DATA GROUP A is not recorded in the diagnostic file because the conditional dump switch is off.

3. Starting with location TABLEX, 100 words of main storage are dumped in the diagnostic file. If printed, the standared octal format is presented.

4. The environment data, control registers X and A, and main storage locations starting with TABLEY through TABLEY + 199 are recorded in the diagnostic file. If printed, the environment data is printed as to status, control registers are printed always in octal format, and the 200 words of main storage, as specified by 'I', are printd in integer format. Since this dump call does not immediately follow an XIF, XAND, or X$OR call, the conditional dump switch does not have an effect on it.

5. The block of data just prior to the present magnetic tape position and having an internal filename of FILEA is read into buffer DUMPB (initialized in (2) by the X$BUFR procedure), and is then recorded in the diagnostic file. If printed, the standard octal format as specified by '0' is presented.

6. No dump is recorded; the conditional dump switch is turned off.

7. The magnetic tape block, whose internal filename is FILEC, is moved backward one block, and then read forward one block. The block of data is read into the main storage location ALPHA, that is initialized by the X$BUFR procedure in (6). The data is then recorded in the diagnostic file and edited in standard octal format.

8. Assume that the current contents of DRDUMP has a value of 500. Beginning at relative word address 500 of mass storage file FILED, 112 words of data are read into the main storage location ALPHA (initialized by the X$BUFR procedure).

9. The file, whose file control table is at BETA, is conditioned to record all subsequent activity at the item level in the diagnostic file. That is, every time a request is made to read an item, the item to which the item handler points is recorded in the diagnostic file.

10. The file control table BETA is conditioned not to record any subsequent activity at the item level.

11. Registers X1 through X11 and A0 are recorded in the diagnostic file and are edited in standard octal format for printing.

4. File Utility Routines (FURPUR)

4.1. INTRODUCTION

In addition to the Executive control statements, there is a set of control statements that call for the file utility routines (FURPUR). When the Executive encounters a FURPUR control statement, it loads the FURPUR processor. FURPUR continues to process control statements until signaled by the Executive that the next statement is not a FURPUR control statement.

4.1.1. Common Information

The operand fields may contain a filename (see Volume 2–2.6.1), an element name (see Volume 2–2.6.4), or a parameter value, depending on the control statement and its use.

If the filename in any parameter is identical to that specified in the immediately preceding parameter, coding a period in the parameter indicates to the FURPUR processor that the same filename is intended. Omitting the filename completely, including the period, indicates to the FURPUR processor that TPF$ is intended (only valid if the parameter normally specifies a file that resides on sector–formatted mass storage).

As with most other system processors, the FURPUR processor automatically assigns any catalogued file that was not assigned when the FURPUR control statement is encountered. In many cases, the FURPUR processor requires exclusive use of a file, and it places user files in the exclusive–use state as necessary to carry out the specified operation. After use, the FURPUR processor automatically returns all files to the assigned status the file had except when the function of the FURPUR control statement was to alter the file status. Temporary files, when specified, must have been assigned by the user.

Table 4–1 summarizes the name and function of each FURPUR control statement.

Table 4-1. Summary of FURPUR Control Statements

Control Statements	Description
@CHG	Changes element name, version name, read key, write key, and mode of a file. (See 4.2.15.)
@CLOSE	Writes two hardware EOF marks on a magnetic tape file and rewinds the tape. (See 4.2.10.)
@COPIN	Copies elements from an element file located on magnetic tape into a program file on sector-formatted mass storage. (See 4.2.2.)
@COPOUT	Copies a program file, or selected elements from a program file, located on sector-formatted mass storage onto a magnetic tape file in element file format. (See 4.2.3.)
@COPY	Copies a file or element from one file to another. (See 4.2.1.)
@CYCLE	Sets the maximum range of absolute F-cycle numbers to be retained for specified files which are listed in the master file directory or sets the maximum number of element cycles to be retained for the specified symbolic element. (See 4.2.16.)
@DELETE	Drops catalogued files or marks elements in a program file as deleted. (See 4.2.7.)
@ENABLE	Clears the disable flags for catalogued files. (See 4.2.17.)
@ERS	Returns to the system all sector-formatted mass storage granules allocated to a file. (See 4.2.6.)
@FIND	Locates an element in a magnetic tape file (file must be in element file format) and positions the file before the element's label block. (See 4.2.13.)
@MARK	Writes two hardware EOF marks on a magnetic tape file and positions the tape between the EOF marks. (See 4.2.9.)
@MOVE	Moves a magnetic tape file forward or backward over a specified number of EOF marks. (See 4.2.4.)
@PACK	Rewrites an entire program file, removing specified types of elements (depending on the options specified) and all elements marked as deleted. (See 4.2.14.)
@PCH	Punches program file elements into 80-column cards. (See 4.2.12.)
@PREP	Creates an entry point table from the preambles of the nondeleted relocatable elements of a program file. (See 4.2.11.)
@PRT	Provides a listing of the master file directory items for catalogued files, information regarding temporary files, the table of contents of a program file, or the text of symbolic element (see 4.2.5). Listings of absolute or relocatable elements may be obtained using the LIST processor (see Volume 4-Section 5).
@REWIND	Rewinds magnetic tape files back to the load point of the first reel. (See 4.2.8.)

In most instances, the meanings of options used in FURPUR control statements vary with the statement. The meanings, however, of the following options are the same for all FURPUR control statements:

Option	Description
A	Process absolute elements
C	Do not exit through ERR$ if an error is encountered. The FURPUR processor would go on to process the next command or parameter field if more than two parameter fields are permitted as in the case of the @DELETE,C control statement. The C option can always be used, even when the discussion of the options specifies 'no options'. This option is assumed for demand usage.
O	Process omnibus elements
R	Process relocatable elements
S	Process symbolic elements

The FURPUR control statements are device dependent as well as file-type dependent. Program files exist only on sector-formatted mass storage, and element files exist only on magnetic tape. Thus, the statement, "the @PCH control statement is used to punch program file elements into 80-column cards" necessarily implies a mass-storage-to-card transfer. If the program file has been copied onto magnetic tape, the @PCH control statement cannot be used to punch elements into cards. The program file elements must be returned to sector-formatted mass storage prior to the attempt to execute the @PCH control statement.

4.1.2. Simultaneous Use of Files

The FURPUR processor, in common with other system processors, such as the Collector, can directly access catalogued program files, even though they have not been assigned to the user's run. The mechanism which the FURPUR processor and the other processors use is the same; that is, a dynamic @ASG (see Volume 2-3.7.1), with the appropriate options, is done using a CSF$ request (see Volume 2-4.10.1.1). These processors return each catalogued file to its original assignment status, using a dynamic @FREE control statement (see Volume 2-3.7.4) with the appropriate options.

FURPUR, like other processors uses the X option (execlusive use) before any file update operation. This is done to make certain that no other runs currently in the system will add or delete elements, or otherwise tamper with the file, while the processor is attempting to carry out its updating function.

If a dynamic @ASG,AX is attempted, from a batch-mode run and another run already has the requested program file assigned, the CSF$ request is held until the file is available. If the user wishes to avoid this condition he should assign the file with AXZ options prior to the FURPUR call.

4.1.3. Multireel Files

The FURPUR processor automatically generates and checks for end-of-reel sentinels.

Commands that write on tape, @COPOUT, @COPY, @MARK, generate an end-of-reel sentinel when the hardware returns an end-of-tape status. TSWAP$ is used to mount the next reel of the file or a blank if none was specified. The end-of-reel sentinel has the form: EOF mark, end-of-reel (14 words with 054 in S1 of the first word), EOF mark, EOF mark.

Commands that read tape, @COPIN, @COPY, @FIND, @MOVE, check the block following each EOF mark for the end-of-reel sentinel. If it is end-of-reel, TSWAP$ is used to allow processing to continue on the next reel. If it is not end-of-reel, the tape is positioned after the EOF mark.

@MOVE and @FIND are restricted with multireel files in that they have no knowledge of reels preceding the one presently in use. They will, however, continue on succeeding reels. @REWIND returns the first reel of the file to the user when it returns control.

4.1.4. Basic File Formats

Figure 4-1 illustrates the relationships of files to each other. The exact formats have been simplified for clarity. The control statements illustrated are control statements that change the format of the files.

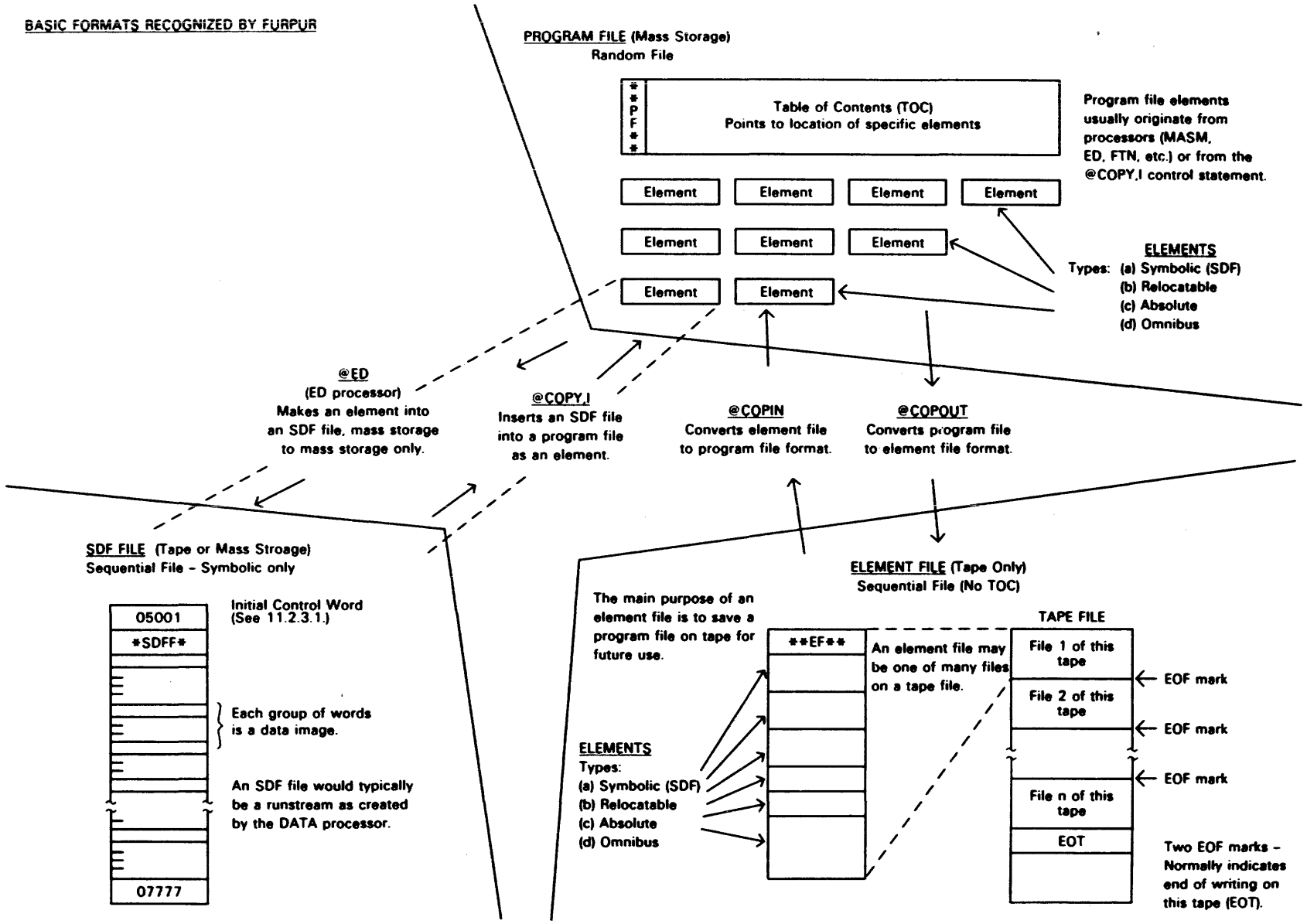


Figure 4-1. FURPUR Control Statements Used to Alter File Formats

4.2. FURPUR CONTROL STATEMENTS

Paragraphs 4.2.1 through 4.2.17 describe the various FURPUR control statements. The most frequently used control statements are presented first and the infrequently used control statements are presented last.

4.2.1. File Copying (@COPY)

Purpose:

Copies a file or element to another file.

All parameters of the @COPY control statement are optional.

Format:

```
@label:COPY,options  name-1,name-2,no.-of-files
```

Parameters:

options	See Table 4-2 for file options and Table 4-3 for element options. See 4.1.1 for additional information on the A, C, O, R, and S options.
name-1	Specifies the input file or element to be copied.
name-2	Specifies the output file into which the input file or element is to be copied.
nbr-of-files	Specifies the number of files to copy; if omitted, one is assumed. When used for tape-to-tape copying of entire files, it specifies the number of input files to copy onto the output tape. When an attempt is made to copy an empty file (two hardware EOFs), the copy operation is immediately terminated regardless of the contents of the nbr-of-files parameter. The input tape remains positioned between the two EOF marks. The number of blocks in each file copied and the number of files copied are indicated in the output listing. When used in conjunction with the B option it specifies the number of 1100 Series FORTRAN files to copy from one file to another. The number of files copied is indicated in the output listing.

Description:

See Volume 2-2.6.1 for additional information on specifying filenames.

When a procedure element is copied the procedure name entries are automatically added to the output file's procedure name table. If a relocatable element is copied, the output file's entry point table is destroyed and the @PREP control statement (see 4.2.11) must be used to recreate it.

Table 4-2. @COPY Control Statement, Options Filenames Specified

Option Character	Description
No option specified	mass-storage-to-mass-storage copying - Overwrite one mass storage file (name-2) with the contents of another mass storage file (name-1), without regard to the file's format. Tape-to-tape copying - Copies one or more files (depending on no.-of-files parameter) from the input tape to the output tape without regard to the file's format. No hardware EOF marks are written (see M option).
A,O,R,S	Copy the elements of the type specified from one program file and add them to another. Both program files must be located on sector-formatted mass storage. Only non-deleted elements are copied. All non-deleted elements of the type specified by the selected options are copied into the output file. Any combination of A, O, R, and S can be used. When the S option is specified, all element cycles are copied.
B	Used with 1100 Series FORTRAN-formatted data files only. Copy the number-of-files (FORTRAN)-specified from the input file to the output file. The input and output files cannot both be on mass storage or both be tape files. When the input file is on mass storage, each software EOF encountered designates the end of a FORTRAN file, and is followed by a hardware EOF when written on tape. When the input file is on tape, each hardware EOF encountered designates the end of a FORTRAN file, and is not maintained when written on mass storage.
F	Copy the contents of one file into another file. Program and element files must not be copied using this option. The input file must be in SDF. Reading of the input file is terminated by the SDF EOF mark. Block sizes for tape files must be 224 words. When the output file is a magnetic tape file, two hardware EOF marks are written following the file and the tape is positioned between the two EOF marks.
G	The G option provides an efficient method of saving and recreating sector-formatted files. Since track size blocks of data are transferred on a @COPY,G operation without regard to format of the contents, the transfer is done relatively quickly and the file's contents are not changed. See 4.3 for format. mass-storage-to-tape copying - The M option may be used to write two EOF marks and position the tape between them after a @COPY,GM operation. Each allocated track of the file, beginning with relative track 0, is written to tape in a 1794 word block. Each 1792 word track is prefixed with two words containing the relative track address, block sequence number and checksum. The @COPY operation is terminated after the highest track written in the file has been written to tape. The first block in the output file is a 28 word label block that contains the file format indicator (COPYG), checksum flag, filename, qualifier, f-cycle, subsystem, unit, highest track written for mass storage, and time and date of the @COPY operations. Tape-to-mass-storage copying - The first two words of each tape block provide the track address into which the block (minus the first two words) is to be written, checksum and block sequence data. If the checksum was present, it is validated. Copying continues until an EOF is encountered. If the tape was created with file information saved in the label, the information will be displayed as follows: QUALIFIER*FILENAME (f-cycle) COPIED ON MM/DD/YY AT HH:MM:SS

Table 4-2. @COPY Control Statement, Options Filenames Specified (continued)

Option Character	Description
I	Used to add an SDF file to a program file as a symbolic element. name-1 - Specifies the input file in SDF. name-2 - Specifies the output file and element name. The SDF file being copied is entered into the program file (located on sector-formatted mass storage) as a symbolic element with an element cycle of 0. Reading of the input file (which may be either tape or sector-formatted mass storage) is terminated by an SDF EOF image. An element created by @COPY,I will retain the image control words (see Volume 2-2.1.4, 2.2.1.2) of the SDF file from which it was created. When the new element is referenced, S3, S4, S5, S6 in the data image control words, may be treated as cycle information by many system processors.
M	The option can be specified only when the output file is a magnetic tape file. mass-storage-to-tape copying - Used with the G option to copy a FASTRAND drum-formatted mass storage file to magnetic tape. Two hardware EOF marks are written on the tape following the file copied, and the tape is positioned between the two EOF marks. (Also applies to no option case). Tape-to-tape copying - Used without other options or with the N option for tape-to-tape copying of one or more files (depending on no.-of-files parameter). If more than one file is being copied, a hardware EOF mark is written on the tape following each file copied except the last, where two hardware EOF marks are written and the tape is positioned between the two.
N	Copy a magnetic tape file containing an abnormal frame count to another magnetic tape file or to a sector-formatted mass storage file. When the output,file is tape, the M option may be used along with the N option to write hardware EOF marks.
P	Used to copy all nondeleted elements from one program file and add them to another. Both program files must be located on sector-formatted mass storage. Can be used in conjunction with the A, O, R, or S options.
V	Copy one file into another file. The input and output file must not both be on magnetic tape or sector-formatted mass storage. mass-storage-to-tape copying - Variable block size is assumed. The first word of the block (containing the block size) is stripped from the block before it is written into the tape file. Tape-to-mass-storage copying - A word containing the block size is prefixed to the block before it is written on sector-formatted mass storage. The input tape file must be terminated by a hardware EOF mark. A copy to or from sector-formatted mass storage always begins in sector 0 and each block starts in a new sector. If the block size is not divisible by 28, the excess words of the last sector contain random data. When used in conjunction with the A, O, P, R, and S options, see Table 4-3.

Table 4-3. @COPY Control Statement, Options Element Names Specified

Option Character	Description
A,O,R,S	Copy the specified element in the input program file and add it to the output program file. The options represent the types of elements to be copied (one or more is needed). The element name can be changed by renaming it in name-2. Both input and output files must be on sector-formatted mass storage. When a symbolic element is being copied, only the element cycle specified or implied in name-1 is transferred to the output file, creating cycle 0 of the element.
V	When used with the A, O, P, R, and S options, copy the specified types of elements in the input program file with the same version name as specified in name-1 to the output program file specified in name-2. When the version name is omitted from name-1, only those elements having a blank version name are copied into the output file. When a version name is given in name-2, it replaces the original version name. When the version name is omitted from name-2, the elements written into the output file retain the version names they had in the input file. The version mask capability is available when the V option is specified (see Table 4-7).

Before doing any copy operation from tape, it may be necessary to execute a @MOVE control statement (see 4.2.4) to position the tape beyond some EOF mark. No @COPY operation will move backward or forward over an EOF mark prior to the start of the copy.

Examples:

In the following examples, tape filenames start with a T, and sector-formatted mass storage filenames start with an F.

```
1.  @COPY          FLAP4.,FLAP5.
2.  @COPY,M        TRAP3.,TRAP6.,9
3.  @COPY,GM       FILLUP.,TANK.
4.  @COPY,P        FLYBY.,FLIGHT.
5.  @COPY,I        FLIP.,FORK.UPT3/INOUT
6.  @COPY,RS       FOR6.,FOR9
7.  @COPY,RS       FIRM.C,FIREUP.A
8.  @COPY,NM       TAP1.,TAP2.
9.  @COPY,F        F1.,F2.
10. @COPY,B        FORT.,TAP2.,3
11. @COPY,RS       FIL1.,FIL2.
12. @COPY,O        F1.,F2.
13. @COPY,AOV      FI./VERS,FO.
14. @COPY,ARSV     FILE1./V**********,FILE2./NEWVERSION
15. @COPY,SRV      F.ELT/VERS,FF.ELTNEW/VERSNEW
```

1. The contents of sector-formatted mass storage files FLAP4 is copied into sector-formatted mass storage file FLAP5 over-writing any previous contents of the FLAP5 file.

2. The nine files which form magnetic tape file TRAP3 are copied into magnetic tape file TRAP6. Each file in output file TRAP6 is separated by EOF marks as directed by the M option. The last file copied into the output file is followed by two EOF marks and the output file is positioned between the two final EOF marks.

3. Copy the contents of sector-formatted mass storage file FILLUP into magnetic tape file TANK. Two EOF marks are written at the end of the output file (TANK) and the tape is positioned between the two EOF marks (M option). Since file TANK is in @COPY,G format, the @FIND and @COPIN control statements cannot be used to access the file; however, @COPY,G format makes more economical use of time and space. The entire file, as it was before this operation was initiated, including all tables of contents and deleted elements, is reproduced when the file is returned to sector-formatted mass storage using a @COPY,G control statement. Do not attempt to merge two program files, each of which were saved on tape using the @COPY,GM control statement because the second file would overlay the first.

4. The nondeleted elements of program file FLYBY are copied into program file FLIGHT.

5. The contents of input file FLIP, which is in SDF, is copied into output file FORK in program file format. Input file FLIP is entered in FORK as an element having UPT3 as its element name and INOUT as its version name. It is set at element cycle 0.

6. The nondeleted relocatable and symbolic elements (R and S options) located in program file FOR6 are copied into program file FOR9.

7. The nondeleted relocatable and symbolic elements with element name C (version name of spaces) in program file FIRM are copied into program file FIREUP as elements with element name A (version name of spaces).

8. One file of magnetic tape file TAP1 is copied onto magnetic tape TAP2. Two EOF marks are written on TAP2 and the tape is positioned between the two. File TAP1 may contain abnormal frame counts.

9. The contents of file F1 is copied onto file F2. File F1 must be an SDF file.

10. The first three FORTRAN files in the file FORT are copied onto tape file TAP2. File FORT must contain at least three FORTRAN files. A hardware EOF mark will follow each of the three files written on TAP2.

11. The nondeleted symbolic and relocatable elements of program file FIL1 are copied into program file FIL2.

12. The nondeleted omnibus elements in program file F1 are copied into program file F2.

13. All nondeleted absolute and omnibus elements in program file F1 having a version name of VERS are copied into program file FO. Element and version names remain unchanged.

14. All nondeleted absolute, relocatable, and symbolic elements in program file FILE1 having a version name beginning with V are copied into program file FILE2 and given a version name of NEWVERSION.

15. The nondeleted relocatable and symbolic elements named ELT in program file F having VERS as a version name are copied into program file FF and given an element name of ELTNEW and version name of VERSNEW.

4.2.2. Copying from Tape to Program Files (@COPIN)

Purpose:

Copies one or more elements from an element file located on magnetic tape into a program file located on sector-formatted mass storage.

All parameters of the @COPIN control statement are optional.

Format:

@label:COPIN,options name-1,name-2

Parameters:

options	See Table 4-4 for file options and Table 4-5 for element options. See 4.1.1 for additional information on the A, C, O, R, and S options.
name-1	Specifies the input element file or input element to be copied.
name-2	Specifies the output program file or output program file and element name. If name-1 is an element name and name-2 is a filename, the element will retain its name in the new file.

Description:

See Volume 2-2.6.1 for additional information on specifying file and element names.

Procedure names are saved, but the entry points were discarded when the program file was converted to an element file with a @COPOUT control statement (see 4.2.3). When a relocatable element is added to a program file, any entry point table that may have existed for the file prior to the execution of the @COPIN control statement is destroyed. The @PREP control statement (see 4.2.11) may be used to recreate the entry point table. If a tape error occurs, only those elements transferred before the error occurred are entered in the program file's table of contents.

Table 4-4. @COPIN Control Statement, Options Filenames Only Specified

Option Character	Description
No option specified	Copies elements from the input magnetic tape file (in element file format) into the program file located on sector-formatted mass storage. The tape file must be positioned at the label block of the first element being copied (use @FIND control statement - see 4.2.13) and continues until a hardware EOF mark is encountered. The element retain the element name they had in the element file.
A,O,R,S	Same operation as if no options were specified except that the A. O, R, and S options can be used to designate the type of elements to be copied. Any combination of A, O, R, and S can be used. All elements of the types specified are transferred. The elements retain the names they had in the element file.
V	Same as for elements (see Table 4-5).

Table 4-5. @COPIN Control Statement, Options Element Names Specified

Option Character	Description
A,O,R,S	One element is copied and inserted into the output program file. The element name remains the same unless renamed in name-2.
V	Used to copy elements having the same version name from an element file on magnetic tape into a program file on sector-formatted mass storage. The input file must be positioned at the label block of the first element to be copied and copying continues until a hardware EOF mark is encountered. name-1 – Specifies an input file, or input file and element version name. name-2 – Specifies an output file, or output file and element version name. When the version name is omitted from name-1, only those elements having a blank version name are copied into the output file. When the version is omitted from name-2, the elements retain the version names they had in the input file. The V option may be used with the A, O, R, and S options to select particular types of elements within version names for copying. The version mask capability is available when the V option is specified on the @COPIN control statement (see Table 4-7).

Examples:

```
1.  @COPIN          HOLDPROG.,PROGRAM.
2.  @COPIN,R        TEMP.ELTA,PF1.
3.  @COPIN,RV       A./B,C.
4.  @COPIN,SV       A.,C.
5.  @COPIN,R        PET.ELT3,REPET.VERSG.
6.  @COPIN,O        T.,F.
7.  @COPIN,SV       TAPE./V**********,FAST.
8.  @COPIN,AV       TAP.OLDABS/VERS********,FAS.
```

1. Element file HOLDPROG located on magnetic tape is copied and reformatted into program file format and added to program file PROGRAM on sector-formatted mass storage.

2. Relocatable element ELTA in element file TEMP is copied into program file PF1. The entry point table of file PF1 is not updated (a @PREP control statement is needed to update the entry point table – see 4.2.11).

3. All relocatable elements in element file A with the version name B are copied into program file C. They retain the version name B in the program file.

4. All symbolic elements in element file A with a blank version name are copied into program file C. These elements added to the program file have a blank version name.

5. The relocatable element in element file PET with the element name ELT3 and a blank version name is copied into program file REPET and given the element name VERSG with a blank version name.

6. All omnibus elements in element file T are copied into program file F.

7. All symbolic elements in element file TAPE having a version name beginning with V are copied into program file FAST.

8. All absolute elements named OLDABS having a version name beginning with VERS are copied into program file FAS. The element and version names remain the same.

4.2.3. Copying Program Files to Tape (@COPOUT)

Purpose:

Copies a program file, or selected elements from a program file, located on sector-formatted mass storage into a magnetic tape file in element file format.

All parameters of the @COPOUT control statement are optional except name-2.

Format:

```
@label:COPOUT,options  name-1,name-2
```

Parameters:

options	See Table 4-6 for file options and Table 4-7 for element options. See 4.1.1 for additional information on the A, C, O, R, and S options.
name-1	Specifies the input program file or element to be copied.
name-2	Specifies the output element file, or output element file and element name.

Table 4-6. @COPOUT Control Statement, Filenames Specified

Option Character	Description
No option specified	All nondeleted elements are written onto the magnetic tape output file in element file format. Two EOF marks are written at the end of the file and the tape is backspaced one EOF mark. Elements retain the element name they had in the program file.
A,O,R,S	All nondeleted elements of the types specified by the options are written onto the magnetic tape output file in element file format. The elements retain the names they had in the program file. Any combination of A, O, R, and S can be used. No EOF mark is written.
V	Same as for elements (see Table 4-7).

Table 4-7. @COPOUT Control Statement, Options Element Names Specified

Option Character	Description
A,O,R,S	All specified element types for the element names given are written into the output magnetic tape file in the element file format. Only nondeleted elements are transferred. If the name-2 element name is different than the name-1 element name, all elements copied have the new name. One or more options must be specified. No EOF mark is written.
V	All nondeleted elements, selected by version name and type, are written onto the magnetic tape output file in element file format. The V option may be used in combination with the A, O, R, and S options. When it is used alone, all element types are considered. name-1 - Specifies an input file, or an input file and element version name. The file must be on sector-formatted mass storage and be in program file format. name-2 - Specifies an output file, or an output file and element version name. If the version name is omitted from name-1, only those elements with a blank version name are considered for copying into the output file. When a version name is given in name-2, it replaces the original version name. When the version name is omitted from name-2, the elements written into the output file retain the names they had in the input file. Version Mask - An * in the version name in name-1 causes the character in the corresponding position in the version names of the elements in the input file to be ignored. For example: TAB./**D********* in name-1 would write all nondeleted elements in file TAB with a D as the third character in their version name into the output file. The V option must be specified when the version mask is used. No EOF mark is written.

Description:

See 4.1.1 for additional information on specifying filenames.

Procedure name entries are saved but relocatable entry points are discarded. Tape files must be in element file format in order to use the @FIND and @COPIN control statements (see 4.2.13 and 4.2.2, respectively).

If either the A, O, R, S, or V option is specified on the @COPOUT control statement, an EOF mark is not written automatically and a final @MARK control statement (see 4.2.9) may, therefore, be necessary.

Example:

```
1.  @COPOUT          PROGRAM.,HOLDPROG.
2.  @COPOUT,ARS      C.,D.
3.  @COPOUT,R        A.,B.
4.  @COPOUT,S        A.B,C.D
5.  @COPOUT,SV       A./B,C.
6.  @COPOUT,AV       A.,C.
7.  @COPOUT,V        A./*B**********,C.
```

1. The contents of program file PROGRAM located on sector-formatted mass storage is copied into magnetic tape file HOLDPROG and reformatted as an element file. Since no options are specified, two EOF marks are written following the output file, and the tape is positioned between the EOF marks.

2. The nondeleted absolute, symbolic, and relocatable elements of program file C located on sector-formatted mass storage are copied into magnetic tape file D and reformatted as an element file. No EOF marks are written following the file (option having been specified).

3. All nondeleted relocatable elements in program file A located on sector-formatted mass storage are copied into magnetic tape file B and reformatted as an element file. No EOF marks are written following the file.

4. Symbolic element B in program file A is copied into magnetic tape file C in element file format and given element name D.

5. All nondeleted symbolic elements in program file A with a version name of B are copied into magnetic tape file C and retain the version name B. File C is in element file format.

6. All nondeleted absolute elements in program file A with a blank version name are copied into magnetic tape file C. File C is in element file format.

7. All nondeleted elements in program file A with a version name containing a B as the second character are copied to magnetic tape file C. The version names are unchanged.

4.2.4. Positioning Tape Files (@MOVE)

Purpose:

Moves a magnetic tape file forward or backward over a specified number of EOF marks.

All parameters in the @MOVE control statement are optional.

Format:

```
@label:MOVE,option  filename,n
```

Parameters:

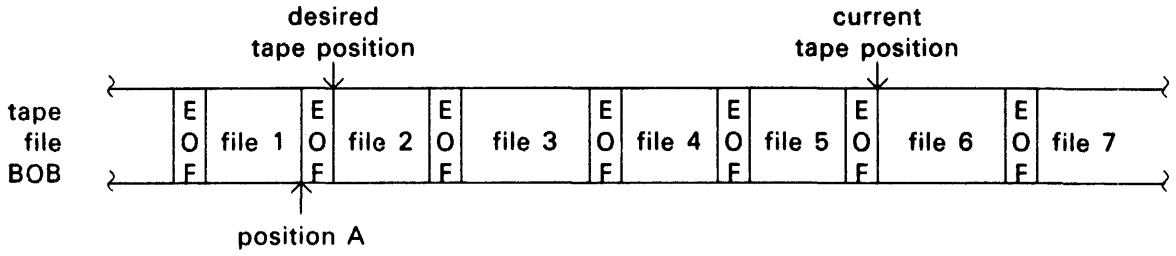
option	The only valid option is B. If specified, tape movement is backward; if omitted, tape movement is forward.
filename	Specifies the name of the tape file.

n — Specifies the number of EOF marks to be skipped. If not specified, 1 will be assumed.

Description:

Tape movement in the forward direction leaves the tape positioned at the start of the file on a multifile reel.

Care must be exercised when moving tape in the backward direction. Assume that tape file BOB is positioned at file 6:



To position the tape to the start of file 2, the following sequence must be executed:

```
@MOVE,B   BOB,5

@MOVE     BOB,1
```

Step 1 moves the tape to position A, and step 2 moves the tape to the start of file 2.

If a @MOVE,B control statement for a multireel tape file encounters the load point of the file it is currently on, a diagnostic message is given and the ERR$ exit is taken.

4.2.5. Listing Files, Elements, and Master File Directory (@PRT)

Purpose:

Obtains a listing of the text of a symbolic element, the table of contents of a program file, information regarding temporary files, or the master file directory items of catalogued files. The control statement does not list absolute or relocatable elements, as this may be done by the LIST processor (see Volume 4 – Section 5).

All parameters in the @PRT control statement are optional.

Format:

```
@label:PRT,options    name-1,name-2...,name-n
```

Parameters:

options — See Table 4-8 for options applicable to files, project-ids, and account numbers and Table 4-9 for element options.

names

Specifies any of the following depending on options indicated:

- the name of a catalogued file in any format
- the name of a program file
- the name of a symbolic element
- the name of a temporary file
- an account number
- a project-id
- removable disk pack-id/disk equipment type
- the number of elements to print

Table 4-8. @PRT Control Options

Option Character	Description
No option specified	When no name fields are specified and the requesting run is privileged, the entire master file directory is displayed alphanumerically by project-id. When the requesting run is nonprivileged, all public files in the master file directory are listed, followed by all files catalogued with the requesting run's project-id. (Read/write keys are not displayed.) The items are sorted first by project-id, then by account number, and then by qualifier and filename. If name fields are specified, elements must be indicated (see Table 4-9).
A,O,R,S	When used in conjunction with the T option, list the table of contents entry for the type(s) of elements specified by the options. The A, O, R, S options may be used with the B, L, and V options, but must be accompanied by the T option.
B	When used in conjunction with the T option, list the table of contents in descending sequence order; i.e., list the most recently inserted elements first. The B option may be used with any combination of A, O, R, S, and V options, but must be accompanied by the T option. Only two names are allowed if the B option is used, and all other names are ignored. The number of elements printed is limited to the number given as name-2. There is no number limit if name-2 is not given.
D	Display the names of all catalogued files currently residing wholly or partially on the named removable disk pack(s). When the requesting run is nonprivileged, all read/write keys and project-ids not matching the run's project-id are replaced by slashes. The files are sorted alphanumerically by project-id, unless the N option is also specified, in which case the files are sorted alphanumerically by account number. Completion of a @PRT,D will require the pack(s) to be mounted. The NAME parameter must specify a pack-id and the disk equipment separated by a slash e.g., PACKID/F14.
F	Display the information from the master file directory for each catalogued file specified. Display the information pertinent to each temporary file specified.

Table 4-8. @PRT Control Options (continued)

Option Character	Description
I	Display the names of all catalogued and temporary files currently assigned to the run. (The files SYS$*LIB$, SYS$*RLIB$ and DIAG$ will be excluded from the list.) Because this feature is primarily for demand runs, the output is displayed in an abbreviated format. No name fields are necessary. Information such as external filenames, internal filenames attached via @USE, and equipment type are displayed for each file in the form: QUAL*FILE(F/C),equipment,assign-options usenames The assign-options possible are defined as: A - assigned with the A option C - assigned with C or U options D - assigned with the D or K options T - assigned as a temporary file X - assigned exclusively If a usename exists but the file is not assigned, the word DUMMY is printed in the equipment field.
L	Used in conjunction with D, N, P, T, or no options or @PRT command (in a demand run) to obtain a complete rather than the abbreviated listing normally produced for demand runs.
N	Display by account number. When the requesting run is privileged, the lack of a name field causes the entire master file directory to be displayed and sorted alphanumerically by account number. When name fields are specified, only the files catalogued with the specified account numbers are displayed. They are displayed in the order specified in the name fields. In nonprivileged mode a name field is unnecessary and if specified, is ignored. In this mode all catalogued files with the requesting run's account number are displayed. They are sorted alphanumerically by project-id. (Read/write keys are not displayed.)
P	Display by project-id. When the requesting run is privileged, the lack of a name field causes the entire master file directory to be displayed and sorted alphanumerically by project-id. When name fields are specified, only the files catalogued with the specified project-id are displayed. They are sorted in the order specified in the name fields. In nonprivileged mode a name field is unnecessary and if specified, is ignored. In this mode all catalogued files with the requesting run's project-id are displayed and sorted alphanumerically by account number. (Read/write keys are not displayed.)
T	Display the table of contents for each specified program file or program file element.
U	Display the current usage of the removable disk pack specified. The name fields are the same as the @PRT,D command. The output consists of the first line of the @PRT,D output, which displays pack-id, tracks available, positions available, and current number of assigns.

Table 4-8. @PRT Control Options (continued)

Option Character	Description
V	When used in conjunction with the T option, list the table of contents for elements with the same version name as specified in the name field. When the version name is omitted from the name field, only those elements having a blank version name are listed. The V option may be used with any combination of A, B, O, R, and S options, but must be accompanied by the T option. The version mask capability is available when the V option is specified on the @PRT,T control statement (see Table 4-7).

Table 4-9. @PRT Control Statement, Options with Elements Specified

Option Character	Description
No option specified	List the text of the specified symbolic elements.
A,O,R,S	Same as for files (see Table 4-8).
B	Same as for files (see Table 4-8).
S	Same as if no options were specified.
T	List the table of contents entry for each element specified.
V	Same as for files (see Table 4-8).

Description:

When the @PRT control statement is used to obtain a display of the master file directory and the requesting run is nonprivileged, all read/write keys are replaced by slashes. All project-ids not matching the requesting run's project-id are also replaced by slashes.

The table of contents information output from the execution of a @PRT,T control statement contains heading information describing the contents of the table of contents. The next write location is always printed. The procedure tables and entry point tables are printed only if no restrictions are placed on the @PRT,T control statement, such as an element name, or the B, A, O, R, S, or V options. Some of the heading information is not self-explanatory. This includes:

Element Table:

DELETE FLAG — An asterisk means entry deleted. No other symbol is used.

TYPE	If the element is symbolic, the processor which created the element is indicated. '-Q' indicates that the ASCII code bit is set in the element table.
DATE AND TIME	Time that element was created or, in some cases, when it was added to this file.
SEQUENCE NO.	The element sequence number is the position of the element in this file (this is sequentially issued) as elements are added to the file.
SIZE-PRE,TEXT	TEXT is the text size in sectors (a sector is 28 words). PRE is the preamble size in sectors (relocatable elements only).
CYCLE WORD	The first field is the maximum number of cycles (cycle limit) to be maintained for the element (see Volume 2-2.6.5). The second field is the most current cycle (absolute). The third field is the number of cycles currently being maintained.
PSRMODE	QTR if element is quarter-word sensitive. THR if element is third-word sensitive. BOTH if element is both quarter and third-word sensitive. SET if element has Arithmetic Fault Compatibility mode bit set. CLR if element has Arithmetic Fault Noninterrupt mode bit set. INS if element is insensitive to Arithmetic Fault handling.
LOCATION (Relative Sector Nbr)	Specifies the sector position of the start of the text.

Procedure Table (Assembler, COBOL, FORTRAN):

DELETE FLAG	An asterisk means entry deleted. No other symbol is used.
LOCATION	Refers to the word position relative to the start of the file.
LINK	The sequence number of the element that contains this procedure name.

Entry Point Table:

NAME	Name of externally defined symbol.
LINK	The sequence number of the element that contains this entry point.

The @PRT,TL control statement from a demand terminal results in the listing at the demand terminal of the table of contents in the format described above. When using the TL options, if an element name is given in addition to the filename, the table of contents is listed for the specified element only.

When the L option is omitted, the @PRT,T control statement from a demand terminal results in the following shortened table of contents format:

type element-name/version(cycle)

where:

type — Indicates the type of element. (See 11.2.1.1.)

Omnibus subtypes are prefixed with 'O-'.

cycles — Indicates the latest element cycle.

Examples:

```
1.  @PRT,T        PROGFILE.
2.  @PRT          PROGFILE.SAM/XYZ
3.  @PRT,P        MERCURY
4.  @PRT,TL       ELEY
5.  @PRT,I
6.  @PRT,D        PAC1/F14
7.  @PRT,F        BASE.
8.  @PRT,TB       F1
9.  @PRT,TS       F1
10. @PRT,TO       F2
11. @PRT,TV       FILE./VERS
12. @PRT,TSV      FAST./V**********
```

1. The table of contents for program file PROGFILE is listed following the format given in Description. The period must follow the filename; otherwise, the specified name is considered to be an element name in TPF$.

2. The most recent cycle of symbolic element SAM, version XYZ, in program file PROGFILE is listed.

3. Information from the master file directory items for all catalogued files whose project-id is MERCURY is listed subject to current system security restrictions. This information is completely labeled to prevent any ambiguities as to the meaning of any entry in the listing. The project-id MERCURY must be tne same as the run's project-id. If not, no listing is generated, unless the run is privileged.

4. A complete table of contents is to be given for element ELEY in TPF$.

5. All files currently assigned to the run are displayed.

6. All catalogued files residing on removable disk pack PAC1 are displayed. PAC1 is an F14 disk pack.

7. Information describing the file BASE is displayed. BASE may be either temporary or catalogued.

8. The element table of contents for program file F1 is listed beginning with the most recently inserted elements.

9. All symbolic elements in the table of contents of program file F1 are listed.

10. All omnibus elements in the table of contents of program file F2 are listed.

11. The table of contents for all elements having a version name of VERS is listed for program file FILE.

12. All symbolic elements with a version name beginning with a V in the table of contents of program file FAST are listed.

4.2.6. Emptying a File (@ERS)

Purpose:

Makes a file available for use as either a program file or an SDF file, with or without releasing allocated sector-formatted mass storage granules. If granule zero is to remain allocated, the first sector will be zero filled.

All parameters in the @ERS control statement are optional.

Format:

```
@label:ERS,options  filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the files to empty.
no-option	Release all granules except those initially reserved.
options	I – Release all granules including initial reserve.
	N – Do not release any granules. If granule zero is allocated, the first sector will be zero filled.

4.2.7. Deleting Files and Elements (@DELETE)

Purpose:

Drops catalogued files or marks elements in program files as deleted.

All parameters in the @DELETE control statement are optional except name-1.

Format:

```
@label:DELETE,options  name-1,name-2,...,name-n
```

Parameters:

names	Specifies the catalogued file or the element to be deleted.
options	See 4.1.1 for additional information on the A, C, O, R, and S options.

- No options apply when deleting a catalogued file.

- Deleting Files

  Each catalogued file specified is marked as dropped. The filename specified may be external or internal.

  When an external filename is specified, the F-cycle must be specified if it is not the latest F-cycle. If the file has read/write keys and is to be assigned to the run by the FURPUR processor, the read/write keys must be specified. The keys may be omitted if the file was assigned prior to calling the FURPUR processor.

  If an internal filename is used, it must have been attached to an external filename by means of an @USE control statement (see Volume 2-3.7.5).

  The file is not actually dropped until all other runs that have the file assigned to them have freed the file. When the file is dropped, the master file directory items are updated. The older F-cycles have their relative F-cycle number increased.

  See Volume 2-2.6.3 for a discussion of file cycles.

  Packs must be mounted for removable disk files being deleted.

- Deleting Elements

  When the A, O, R, and S options are specified, an element of that type in a program file is marked deleted. Each entry in the operand field names the element and the program file that contains it. Any combination of A, O, R, and S options may be used, but at least one must be specified.

  Including a cycle number for a symbolic element is illegal. All cycles of the element must be deleted. All procedure names associated with a deleted procedure element are marked as deleted. The entry point table is destroyed if a relocatable element is deleted. An @PACK control statement (see 4.2.14) may be used to reclaim the physical space occupied by deleted elements.

  When the V option is specified, all elements with the same version name as that specified in the name field are deleted. The V option may be used with one or more of the A, O, R, and S options to select types of elements for deletion.

  The version mask capability is available when the V option is specified on the @DELETE control statement (see Table 4-7).

Description:

The effect of a @DELETE control statement on a catalogued file is the same as the sequence:

```
@ASG,AYD    FILEA
@FREE,D     FILEA
```

Examples:

```
1.  @DELETE,S      F.ELT1/VERS,F1.ELTY
2.  @DELETE        FLAP.,TARE5.,ZEBRA4.,BAKER.
3.  @DELETE,OV     F1./**********
4.  @DELETE,RSV    F2./VERS
5.  @DELETE,V      FAST./V**********
```

1. The symbolic elements ELT1/VERS in program file F and ELTY in program F1 are marked as deleted. Any associated procedure names are also marked as deleted.

2. Relative F-cycle -0 of catalogued files FLAP, TARE5, ZEBRA4, and BAKER is dropped from the master file directory (the files are decatalogued).

3. All omnibus elements in program file F1 are marked as deleted.

4. All relocatable and symbolic elements having a version name of VERS in program file F2 are marked as deleted.

5. All elements having a version name beginning with V in program file FAST are marked as deleted.

4.2.8. Rewinding Tape Files (@REWIND)

Purpose:

Rewinds magnetic tape files back to the load point of the first reel.

All parameters in the @REWIND control statement are optional.

Format:

```
@label:REWIND,options  filename-1,filename-2,...,filename-n
```

Parameters:

options	The C (see 4.1.1) and I options are the only valid options. If the I option is specified, the tape file is rewound with interlock; if omitted, the tape file is rewound without interlock.
filenames	Specifies the tape files to be rewound.

4.2.9. Marking an EOF on Tape (@MARK)

Purpose:

Writes two hardware EOF marks on a magnetic file and leaves the tape positioned between them. Some FURPUR control statements do an automatic @MARK or can be made to do a @MARK by specifying the M option.

All parameters in the @MARK control statement are optional.

Format:

```
@label:MARK  filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the tape files on which hardware EOF marks are to be written.

4.2.10. Closing Tape Files (@CLOSE)

Purpose:

Writes two hardware end-of-file (EOF) marks on a magnetic tape file and then rewinds it.

All parameters in the @CLOSE control statement are optional except filename-1.

Format:

```
@label:CLOSE,options  filename-1,filename-2,...,filename-n
```

Parameters:

options	The C (see 4.1.1) and I options are the only valid options. If the I option is specified, the tape is rewound with interlock; if omitted, the tape is rewound without interlock.
filenames	Specifies the tape files to be closed.

4.2.11. Entry Point Table Creation (@PREP)

Purpose:

Creates an entry point table from the preambles of the nondeleted relocatable elements of a program file.

All parameters in the @PREP control statement are optional.

Format:

```
@label:PREP  filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the program files for which entry point tables are to be created.

Description:

If a previous entry point table existed, it is destroyed prior to creating a new one. Note that whenever a relocatable element is added to or deleted from a file, any existing entry point table is destroyed.

4.2.12. Punching Program File Elements (@PCH)

Purpose:

Punches program file elements into 80-column cards.

All parameters in the @PCH control statement are optional except eltname.

Format:

@label:PCH,options eltname,seq-char

Parameters:

options

See 4.1.1 for additional information on the A, C, O, R, and S options. The function of the A, O, R, and S options is as follows:

A,O,R,S - Specifies the type of element to be punched. Any combination of A, O, R, or S may be used, but at least one must be given.

The following options may be used only in conjunction with the S option (see Volume 4-2.1.4.1 and 4-2.1.4.2):

G - Produce punched card output containing compressed symbolic images

H - Punch sequence number into columns 73-80 of each image. Seq-char must contain an alphabetic sequence of from one to three characters. The characters are left-adjusted and overlay columns 73-75.

J - Compress input images and sequence output cards in columns 73-80.

The G and J options may not both be specified in the same control statement.

eltname

Specifies element to be punched.

seq-char

Specifies alphabetic sequencing characters when the H option is selected.

Description:

See 1.4 for information on how to specify element names.

The FURPUR processor ensures that the elements punched contain the control cards needed to reinsert them into the same program file, or a program file with the same name in a later run. The first card of a procedure element is a @PDP,I control card (see Section 8). For all other elements, it is an @ELT,I card (see Section 5). The filename on the control card is the name of the file from which the element was punched.

Relocatable and absolute elements are automatically (without special option) sequenced in columns 79-80. Sequencing starts with AA and ends with ZZ (starts over with AA if necessary). If the H option

is specified, symbolic images are sequenced in columns 76–80. The card sequence will be 100 apart and is preceded by the designated alphabetic string given in the seq-char field on the @PCH control statement.

The punched output is either preceded by a properly formatted @ELT,I card or a @PDP,I card. These cards produced by @PCH have the same element name as the @PCH control statement. Thus, if the file containing the element that was punched is assigned to a subsequent run, all that is necessary to reintroduce the element is to include the @PCH-produced cards in the run stream.

Examples:

```
1.  @PCH,S        UPDATE.RUNPROG
2.  @PCH,SRJH     A.B,XYZ
```

1. Symbolic element RUNPROG in program file UPDATE is punched onto 80-column cards, one image per card.

2. Symbolic element B of program file A is punched in 80-column cards. The input images are sequenced in columns 76–80. The identification sequence is punched in columns 73–75. The input images are also compressed.

   Relocatable element B of program file A is also punched. The text has been previously sequenced, and the FURPUR processor sequences the preamble.

   See Volume 4-2.1.4.2 for a discussion of compressed symbolic elements.

4.2.13. Positioning within Element Files (@FIND)

Purpose:

Locates an element in a magnetic tape file (file must be in element file format) and positions the tape immediately preceding the element's label block.

All parameters in the @FIND control statements are required except label and options.

Format:

```
@label:FIND,options  eltname
```

Parameters:

options	One and only one of the options A, O, R, or S must be used to specify the type of element. See 4.1.1 for additional information on these options.
eltname	Specifies the file and the element to be located.

Description:

The search is made forward until either the element is found or an EOF mark is encountered. When the EOF mark is encountered, the tape is backspaced to the previous EOF mark (or load point, whichever is encountered first) and the search is repeated. If no find is made, the error exit is taken when the EOF mark is encountered.

Normally, the @FIND control statement is used just prior to a @COPIN control statement (see 4.2.2) requesting that the element just located (or all elements up to the EOF mark) be inserted into a program file located on sector-formatted mass storage. It is also required before calling a language or systems processor which is to use SIR$ to read the element from tape (specified as the SI field).

Care must be exercised when doing a @FIND operation on other than the first reel of a multireel file. If an EOF is encountered prior to locating the desired element the reel is backspaced only to the load point, not to the EOF which is located on a previous reel.

4.2.14. Removal of Deleted Elements (@PACK)

Purpose:

Rewrites an entire program file, removing specified types of elements (depending on the options specified) and all elements marked as deleted. @COPOUT and @COPY,P control statements (see 4.2.3 and 4.2.1, respectively) have the same effect on output files since they do not copy deleted elements. Mass storage space which is no longer needed is returned to the system.

All parameters in the @PACK control statement are optional.

Format:

```
@label:PACK,options  filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the program files to be written.
options	See Table 4-10. See 4.1.1 for additional information on the A, C, O, R, and S options.

Description:

Any combination of A, O, R, and S options may be specified. If no options are specified, then all deleted elements are removed.

Normal Termination:

If the file is position granularity, 'FILE SIZE' is the number of positions required to contain the program file.

If the file is track granularity, 'TEXT' is the number of tracks required to contain the element text, and 'TOC' is the number of tracks required to contain the table of contents.

The number and type of each element is printed at termination.

Abnormal Termination:

If the termination was not caused by an error (i.e., system crash, or @@X T), the @PACK can most likely be completed by repeating the @PACK command with the same options.

Table 4-10. @PACK Control Options

Option Character	Description
no option	Remove all deleted elements and release all unused granules except those granules initially reserved. (see I-option)
A	Remove all elements except nondeleted absolute elements.
D	Used in conjunction with M option to inhibit creation of an entry point table. Ignored if no M option is specified.
I	Release all unused granules including initial reserve.
M	Creates a minimum size table of contents by starting each program file table in the sector following the previous not empty program file table. The element text is started in the first track not used for the table of contents (see Section 6). An entry point table is created unless the D option is specified. In most cases, this entry point table can be larger than if a @PREP followed a @PACK without the M option. After a @PACK,M, there is no room to expand the table of contents, and new elements cannot be added. To expand the table of contents, by adding or updating an element or creating an entry point table, it will be necessary to transfer the elements to another file. @COPY,P or @COPOUT may be used for the transfer. Alternatively, existing elements can be deleted followed by a @PACK without the M option.
N	Do not release any granules. If granule zero is allocated and there are no nondeleted elements, the first sector will be zero filled.
O	Remove all elements except nondeleted omnibus elements.
P	Cause an entry point table to be created, as if a @PREP command immediately followed the @PACK command. The P option is ignored if the M option is specified.
R	Remove all elements except nondeleted relocatable elements.
S	Remove all elements except nondeleted symbolic elements.

4.2.15. Changing Element and Version Names, File Keys and Modes

The @CHG control statement described in 4.2.15.1, discusses how to change catalogued files, keys, and modes; and in 4.2.15.2, discusses how to change program file element and version names. Some examples of @CHG control statements are given in 4.2.15.3.

4.2.15.1. Changing Catalogued Files, Keys and Modes

Purpose:

Changes catalogued mass storage files, keys, and modes.

All parameters in the @CHG control statement are optional except name-1.

Format:

@label:CHG,options name-1,name-2

Parameters:

options — The options are:

- P – Set public mode
- Q – Set private mode
- V – Set read-only mode, clear write-only mode
- W – Set write-only mode, clear read-only mode
- Z – Clear read only and/or write-only modes (must not be used in conjunction with V and W options)

name-1 — Specifies the file to be changed

name-2 — Specifies the same file as name-1 with new or changed keys

Description:

One F-cycle series exists for each set of files with the same filenames. Each catalogued file belongs to only one F-cycle series. Read/write keys, if any, are the same for all members of the series. The master file directory contains a lead item for each F-cycle series that lists the read/write keys for the series and points to a main item for each member of the series. The read-only mode and write-only mode indicators are kept in the main item for that member.

The @CHG control statement may be used to perform the following functions related to catalogued files:

1. Change the read/write keys for all files of a given F-cycle series.
2. Remove or set read-only or write-only modes on a file.
3. Set public or private mode on a file.

If an F-cycle series contains only one member, 1. is equivalent to changing the keys for a file.

Although the functions performed by the @CHG control statement do not include reading or writing in text areas of the files named, read/write keys, if the files have any, are required in order for @CHG to modify their master file directory items. This means that the filename on the first @ASG control statement given to the Executive must include the keys if an external name is used. If an internal name is used, it must be associated by an @USE control statement still in effect that includes the keys. FURPUR performs the initial assignment, if the user has not assigned the file. In this case, the same rules apply to the name furnished on the @CHG control statement as for the @ASG control statement furnished by the user.

If the file being changed is assigned to another run the @CHG will be executed but the actual changes will not be made to the MFD item until the file is no longer assigned to another run.

4.2.15.2. Changing Program File Element and Version Names

Purpose:

Changes program file element and version names.

All parameters in the @CHG control statement are optional.

Format:

```
@label:CHG,options  eltname-1,eltname-2
```

Parameters:

options	The A, C, O, R, S, and V options are the only valid options for this statement (see 4.1.1).
eltname-1	Specifies the program file element.
eltname-2	Specifies the same program file and the new element and version names.

Description:

One or more of the A, O, R, and S options must be specified; only the elements of types specified by options will have their names changed. Element cycles may not be specified.

This operation can also be performed during a @COPIN (see 4.2.2), @COPOUT (see 4.2.3), or a @COPY,ADRS (see 4.2.1) control statement.

The V option, when used for element version changing, must be accompanied by one or more of the A, O, R, and S options. This option allows the changing of element/version names for all elements which have the same version name as that specified in name-1. Only the element types specified by the options will be changed.

The version mask capability is available when the V option is specified on the @CHG control statement (see Table 4-7).

4.2.15.3. @CHG Control Statement Examples

The following examples illustrate the operation of the @CHG control statement.

```
1.  @CHG,S        A.B/F4,A.GOTO/A1
2.  @CHG,ARS      UP.PACK,UP.PACK/VER3
3.  @CHG,R        IN.PUT/A,IN.PUT/F
4.  @CHG,A        OUT.PUT/G,OUT.GO/G
5.  @CHG,V        FILE/KEY1/KEY2.
6.  @CHG          FILE1/KEY1/KEY2.,FILE1/KEYA.
7.  @CHG,ORS      F.ELT,F.NEWELT
8.  @CHG,RSV      F1./VERS,F1./NEWVERS
9.  @CHG,AV       F2./V**********,F2.
```

1. Changes the element and version names of symbolic element B, version F4 of program file A to element name GOTO, version A1.

2. Assigns the version name VER3 to the absolute, relocatable and symbolic elements named PACK in program file UP.

3. Changes the version name of relocatable element PUT in program file IN from A to F.

4. Changes the element name of the absolute element PUT in program file OUT to GO. The version is not altered.

5. Changes the mode of catalogued file FILE from its present mode to read-only mode.

6. Changes the read key of catalogued file FILE1 from KEY1 to KEYA and deletes the write key.

7. Change the omnibus, relocatable, and symbolic elements named ELT in program file F to element name NEWELT.

8. All relocatable and symbolic elements having a version name of VERS in program file F1 are given the version name of NEWVERS. The element names remain unaltered.

9. All absolute elements having a version name beginning with V in program file F2 are given the new version name of blanks. The element names remain unaltered.

4.2.16. Altering Cycle Retention Limit (@CYCLE)

Purpose:

Sets the maximum range of absolute F-cycle numbers to be retained for a catalogued file (see Volume 2-2.6.3) or the maximum number of element cycles for a program file symbolic element (see Volume 2-2.6.5). When n is omitted, cycle information regarding the current F-cycle series is displayed.

All parameters of the @CYCLE control statement are optional.

Format:

```
@label:CYCLE  name,n
```

FILES

Parameters:

name	Specifies an F-cycle series for which the cycle range is to be changed.
n	Specifies the maximum range of F-cycles to be retained. If omitted, cycle information is displayed regarding the current F-cycle series.

Description:

When a catalogued file is specified, the @CYCLE control statement sets the maximum range of F-cycles for the filename. The file specified must be in the master file directory. If n is 0, the F-cycle series is deleted. If n specifies a new maximum less than the current range of F-cycles being retained, enough F-cycles of the file set (starting with the oldest cycle) are deleted to satisfy the new range.

ELEMENTS

Parameters:

name	Specifies a program file symbolic element whose cycle limit is to be changed.
n	Specifies the maximum number of element cycles to be retained.

Description:

When a program file element is named, the @CYCLE control statement sets the maximum number of element cycles. If n specifies a new maximum less than the current number of element cycles being retained, a new element with the same name is created and some data image control words may be changed and some images may not be transferred at all.

Any image deleted in a cycle number less than the lowest cycle number being retained, because of the new lower maximum number of cycles, is not transferred to the new element. Any images added but not deleted in a cycle number less than the lowest cycle number being retained are transferred; but S6 of each data image control word is changed to reflect the new lowest cycle number retained. All other nondeleted and deleted images are transferred to the new element.

Examples:

```
1.  @CYCLE      Q*A.B,2
2.  @CYCLE      Q*D.,2
3.  @CYCLE      Q*D.
```

1. Assume that symbolic element B in program file Q*A consists of element cycles 5, 6, 7, and 8. Since the new limit is two cycles, a new element B is created consisting of cycles 7 and 8.

2. Assume that the master file directory entry for file Q*D indicates that four absolute F-cycles 18, 15, 14, and 12 of the file exist. Since the new limit is two, absolute F-cycles 15, 14, and 12 are deleted. The limit is considered to be the range starting from the highest current absolute F-cycle number.

3. F-cycle information is displayed regarding the catalogued file Q*D.

4.2.17. Enabling Files Disabled Due to Malfunctions (@ENABLE)

Purpose:

Resets (removes) the disable flag for catalogued files.

All parameters in the @ENABLE control statement are optional.

Format:

```
@label:ENABLE  filename-1,filename-2,...,filename-n
```

Parameters:

filenames	Specifies the catalogued files to be enabled.

Description:

If the specified file is not disabled, a message to this effect is printed on the listing (normal exit is taken). Keys, if any, must be specified.

4.3. FURPUR FILE FORMAT COPY,G

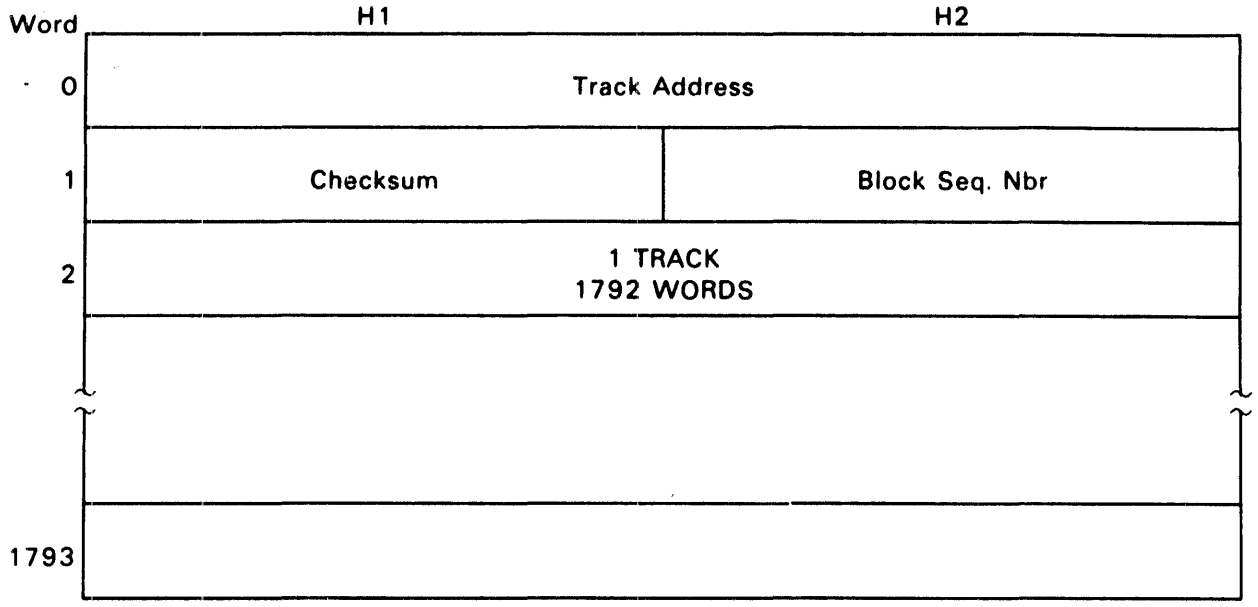
A 28-word label block is written to tape prior to copying the file's contents giving details of the copied file and the time when the copy was made.

Format:

Word	
0	COPYG△
1	BLKSEQ
2 3	Qualifier (LJSF)
4 5	Filename (LJSF)
6	F-cycle
7	Date MMDDYY
8	Time HHMMSS
9	Equipment Code
10	Highest Track Written
~	~
28	

Subsequent blocks are 1794 words in length and consist of a mass storage track (1792 words) preceded by a 2-word header containing the track address and checksum and block sequence number.

Format:



The end-of-file and end-of-reel conditions conform to the same formats described for COPOUT.

5. ELT Processor

5.1. INTRODUCTION

This section describes the ELT processor and the @END control statement (see 5.2.1), which is used with the ELT and DATA (see Section 6) processors.

The ELT processor is used to introduce an element into a particular program file or make corrections to a symbolic element in a program file from within the runstream.

5.2. @ELT FORMAT

Purpose:

Introduces an element into a particular program file or makes corrections to a symbolic element in a program file from the runstream. The @ELT control statement is used to call the ELT processor and it must precede the element or correction images in the runstream. With the exception of the @ELT,D control statement, the ELT processor is terminated by the first nontransparent control statement encountered in the runstream. The ELT,D processor is terminated by an @END control statement (see 5.2.1) whose sentinel matches the sentinel on the @ELT,D control statement. Any control statements, with the exception of the @FIN and @ADD,D statements (see Volume 2-3.4.2 and 2-3.10.1, respectively), appearing between the @ELT,D control statement and the @END control statement are treated as data.

All parameters in the @ELT control statement are optional except eltname-1.

Format:

@label:ELT,options eltname-1,eltname-2,sentinel,time-and-date,bank- information

Parameters:

options — See Table 5-1. All of the source input/output routine (SIR$) options contained in Table 1-2 may be used. The A, O, R, and S options (element type options) identify the element type, while the I, L, and U options principally outline element (image handling) options. The D option is used to insert control statements into a symbolic element. Those elements identified as type S are considered symbolic elements and also may be corrected by using the correction statements (see 1.2).

The S option is assumed when the element type options are not specified. The L option is assumed when the image handling options are omitted and no eltname-2 is given.

eltname-1 — Specifies the input element. With the I option this parameter specifies the new element being inserted into the program file. With the U option this parameter specifies both the symbolic input and output elements.

eltname-2 — Specifies the new output element to be generated. Not used with I. May be specified with U option to retain all element cycles up to cycle specified in a new SO element.

sentinel — Specifies, when the @ELT,D control statement is used, the character code which terminates the flow of data images into the element being created. This parameter may consist of one to six characters and must agree exactly with the sentinel code appearing on the @END control statement (see 5.2.1) used to terminate the ELT processor.

time-and-date — Specifies time and date element was created in TDATE$ format (see Volume 2-4.5.2) shifted circularly 18 bits. Applies to absolute and relocatable elements only. This parameter is optional.

bank-information — Applies to absolute elements only and is required. Contains the same information as absolute element table item word 6 (see Figure 11-3).

Table 5-1. @ELT Control Statement, Options

Option Character	Description
	Element Type Options
A	Identifies element as an absolute element; used only with the I option.
O	Images following the @ELT are inserted as they appear in the run stream into an omnibus element. The element is not formatted by ELT. Applies only with the I option.
R	Identifies element as a relocatable element, used only with the I option.
S	Identifies element as a symbolic element. This option is assumed when no element type option is specified.
	Image Handling Options
D	Indicates that the symbolic input images following the @ELT control statement may include control statements which are to be transferred as data. All control statements are transferred until a @FIN or @END (with matching sentinel) control statement is encountered (see Volume 2-3.4.2 and 5.2.1, respectively).
L	Requests a listing of the complete symbolic element. The listing provides line numbers, cycle information and identification of the newly added and deleted images. This option may be specified for absolute, omnibus or relocatable elements to furnish an unedited listing of the images as they appear in the runstream. This option is assumed when the I, L and U options are omitted and no eltname-2 is given.
X	Take error exit (ERR$ see Volume 2-4.3.2.2) upon occurrence of an error.

Description:

The ELT,D control statement allows insertion of control statements into a program file as elements which may represent @RUN and @ADD runstreams (see Volume 2-3.4.1 and 2-3.10.1, respectively) that can be called later by the @ADD or @START control statements (see Volume 2-3.10.1 and 2-3.4.3, respectively).

When an element is punched by the FURPUR processor (see Section 4), the element is always preceded by an @ELT control statement. The filename punched into the @ELT control statement is the name of the file from which the element was punched. These decks can simply become part of the input to subsequent runs. If the element is to be added to a file other that the one from which it was punched, the filename on the @ELT statement must be changed.

The @ELT statement generated by FURPUR when an element is punched also contains information in specification fields 4 and 5. Field 4 contains the time and date the element was created from word 9 of the Program File table item (see Figure 11-3). Field 5 applies only to absolute elements and contains bank information from word 6 of the absolute element table item. If field 4 is not present the current time and date is used when the element is added to a program file. Field 5 must be present when an absolute element is to be added.

If the H, I, K, and L options are used (see Table 1-2) ELT will identify any column 73-80 sequence number errors by attaching a '#' (pound symbol) to the line number out of sequence.

Examples:

```
1.  @ELT,U          PF1.ELEMENT1
2.  @ELT            PF1.ELEMENT1,PF2.ELEMENT2
3.  @ELT,L          PF1.ELEMENT2
4.  @ELT            PF1.ELEMENT2
5.  @ELT,I          PF1.ELEMENT3
6.  @ELT,IA         ESCP*TPF$.ELT,,,106256042410,004541002746
7.  @ELT,IR         PF1.ELEMENT5
8.  @ELT,IS         PF1.ELEMENT6
9.  @ELT,ID         PF1.ELEMENT7,,STOP
10. @ELT,U          TPF$.A,TPF$.B
11. @ELT,IOL        TPF$.OMNI
```

1. The correction images following this control statement update ELEMENT1 of program file PF1. The updated element replaces the old ELEMENT1 in PF1. Since the element type is not specified, the S option is assumed, and the element is considered to be a symbolic element.

2. The correction images following this control statement are applied to ELEMENT1 of program file PF1 to produce a new symbolic element (ELEMENT2) in program file PF2. ELEMENT1 remains unchanged.

3. This control statement lists ELEMENT2 of program file PF1.

4. The correction images following this control statement are applied to ELEMENT2 of program file PF1. The new symbolic element is listed but no new element is produced because the U option is omitted. ELEMENT2 remains unchanged.

5. The images following this control statement are inserted as a new symbolic element (ELEMENT3) in program file PF1.

6. The images following this control statement are inserted as a new absolute element (ELT) in program file TPF$. The time and date of the element will be 10:00:14 and 20 APR 72. The

I-bank length of this element is 2401 decimal words and the D-bank length is 1510 decimal words.

7. The images following this control statement are inserted as a new relocatable element (ELEMENT5) in program file PF1. Since there is no time and date specified the current time and date is used.

8. The images following this control statement are inserted as a new symbolic element (ELEMENT6) in program file PF1.

9. The images following this control statement are inserted as a new data element (ELEMENT7) in program file PF1. The data stream is terminated when an @END control statement having the matching sentinel STOP is encountered.

10. The correction images following the control statement are applied to element A in program file TPF$ to produce the next higher element cycle which then becomes the new symbolic element B. Element A remains unchanged. Previous element cycles are retained in element B.

11. The images following the @ELT are inserted as a new omnibus element (OMNI) in program file TPF$. The images are listed as they appear in the runstream.

5.2.1. Input Termination Sentinel (@END)

Purpose:

Marks the end of a data file or element. It follows the data images introduced by either an @ELT,D or @DATA controls statement (see Section 6).

The sentinel parameter is optional.

Format:

```
@END    sentinel
```

Parameters:

sentinel — A one- to six-character code corresponding exactly to the sentinel contained in the @DATA or @ELT,D control statement introducing the data images. The end of the file or element is determined when the character string in this parameter matches the character string of the sentinel parameter specified in the associated @DATA or @ELT,D control statement.

Description:

@END control statements cannot have labels and cannot be continued. The @END control statement must be coded exactly as shown (punched into first four columns of the card).

Examples:

```
@END FINISH
```

When the @END control statement is encountered, it ends the data file or element introduced by the @DATA or @ELT,D control statement that has its sentinel parameter coded FINISH.

6. Data Processor

6.1. INTRODUCTION

This section describes the DATA processor. The DATA processor is used to create and update System Data Format (SDF) files from within the runstream.

The @END statement (see 5.2.1) is used in conjunction with the DATA processor.

6.2. @DATA FORMAT

Purpose:

Creates, updates and lists SDF files from the control stream. DATA processor operations is terminated by an @END control statement (see 5.2.1) whose sentinel matches the sentinel in the @DATA control statement.

The @DATA control statement is used to call the DATA processor and it must precede the data or correction images in the runstream. Any control statement, with the exception of the @FIN and @ADD,D control statements (see Volume 2-3.4.2 and 2-3.10.1, respectively) appearing between the @DATA control statement and the @END control statement is treated as data.

All parameters in the @DATA controls statement are optional except filename-1.

Format:

@label:DATA,options filename-1,filename-2,sentinel

Parameters:

options

See Table 6-1. Symbolic input/output routine (SIR$) options (see Table 1-2), except the U option, also apply.

If the I option is omitted, but both filename-1 and filename-2 parameters are specified, the data images following the @DATA control statement are interpreted as corrections to filename-1. A new updated file identified by filename-2 is generated.

If the options and filename-2 parameters are omitted, the L option is assumed. Filename-1 is listed but no new file is generated.

filename-1 — Specifies the file to which the data images and correction images in the runstream apply. The file must be catalogued or assigned to the run.

filename-2 — Specifies the updated file to be generated. The file must be catalogued or assigned to the run.

sentinel — Specifies a character code of one to six characters used for comparison purposes in determining the proper terminating @END control statement (see 5.2.1) for the data mode.

Table 6-1. @DATA Control Statement With Options

Option Character	Description
L	Generates a complete listing of the file. This includes sequential item numbers which are used when making corrections to the file and identification of added and deleted images. If this option and filename-1 are the only parameters specified, filename-1 is listed. This option is assumed when the I, L, and U options are omitted and no filename-2 is given.
U	The U option is used only to update from relative F-cycle 0 to relative F-cycle +1. Only filename-1 can be specified. Relative F-cycle +1 of the file must be assigned prior to implicitly referencing it by means of the U option on the @DATA control statement (see Volume 2-2.6.3).
X	Take error exit (ERR$ - see 4.3.2.2) upon occurrence of an error.

Description:

The difference between the operation of the DATA processor and the @FILE control statement (see Volume 2-3.8.1) is that the DATA processor handles data as it is presented in the runstream at run time, whereas the @FILE control statement builds the file as the data is being initially input into the system. In short, the DATA processor operates identically to a language processor control statement. The file built by the DATA processor is in System Data Format (SDF) (see 11.2.3).

The DATA processor allows the user to build data files which are an entire or partial runstream. These files can then be called on by the @START control statement (see Volume 2-3.4.3) to start an independent run or by the @ADD control statement (see Volume 2-3.10.1) for inclusion in a current or subsequent run. The DATA processor enables the user to make corrections to an independent runstream and then start it using the @START control statement, or make corrections to a partial runstream and add it to the run using the @ADD control statement. The data processor can also be used as a convenient means of generating and maintaining a user's data file rather than a control stream type file.

@DATA does not write a hardware tape mark after writing a file out to tape. The user should do a @MARK after @DATA has completed (see 4.2.9).

Examples:

```
1.  @DATA          FILEA,FILEX
2.  @DATA,I        FILEB
3.  @DATA,L        FILED
4.  @DATA          FILEY
5.  @DATA,IL       FILEZ
6.. @ASG,C         FN(+1)
    @DATA,UW       FN
```

1. The images following this control statement provide the corrections for FILEA. The updated version of this file is stored into the newly created file FILEX.

2. The images following this control statement are inserted into FILEB.

3. The images following this control statement are applied as corrections to FILED. FILED is listed, but a new file is not created.

4. Because the options and filename-2 parameters are omitted, the L option is assumed, and a complete listing is provided for FILEY.

5. The images following this control statement are inserted into FILEZ and listed.

6. The correction images following the data statement are applied to file FN to create relative F-cycle +1 of file FN. The correction images are listed.

7. Text Editor (ED) Processor

7.1. INTRODUCTION

This section describes the ED processor which permits the user to manipulate the text of a symbolic file or element.

7.2. @ED PROCESSOR CALL STATEMENT FORMAT

Purpose:

To invoke the Text Editor (ED processor) and specify its input, output, and operation modes.

All parameters of the @ED control statement are optional.

Format:

```
@label:ED,options   name-1,name-2
```

Paramters:

options	See Table 7-1.
names	Specifies input or output files or elements (see Table 7-1).

Description:

Table 7-1 lists the available options, the input/output files as specified by name-1/name-2, and functions. Unless otherwise specified, name-1 and name-2 may be either files or elements.

If name-1 is omitted, the name of an element in TPF$ will be solicited. If no options of the set C, I, R, U are given, the C option will be assumed.

Table 7-1. @ED Control Statement, Options

Option Character	Name-1	Name-2	Description
Not I, R or U	Input	Output	Input is taken from name-1 and the resultant text is placed in name-2.
Not I, R or U	Input	None	R option assumed for symbolic element. U option assumed for data file.
A	Input	Output	Attempt auto recovery - see AUTO command (Table 7-2).
B	Input	Output	Batch mode when using a demand terminal - the ED processor run will not solicit input from user (see ON/OFF TRDINP).
C	Input or Output	Ignored	Enter input mode if element does not exist. Otherwise, assume U option.
D	Input	Output	Demand mode when using a batch terminal - output listing of the ED processor run will contain solicitation messages.
E	Input	Output	Set EOF mode on at start of edit (see ON/OFF commands).
I	Output	Ignored	Initial insertion of symbolic input from the runstream which causes the ED processor to enter the input mode. The images following the @ED control statement are inserted into the file named in name-1. The I option takes precedence over the R or U option. If the I option is specified or if two fields are specified, then on entry to the text editor a check is made on the output file (i.e., field 1 if I option; field 2 if two fields specified). If the output is a data file and the user specified an element in that file, then the ED processor errors off after printing the error: FILE NOT PROGRAM FILE FORMAT If the output file is a program file and the user did not specify an an element in that file, then the ED processor prints a warning message: WARNING: ON EXIT OUTPUT FILE WILL BECOME A DATA FILE
L	Input	Output	Print all lines following the @ED control statement. The lines printed are indented and preceded by four asterisks.
N	Input	Output	Suppresses printing of changed, found, or relocated lines. This option serves the same purpose as the ON BRIEF command (see Table 7-2).
P	Input	Output	Output file will be Fieldata
Q	Input	Output	Output file will be ASCII.
R	Input	Ignored	Input is taken from name-1 of the @ED control statement; no output text is produced (read-only mode). The UP command should be used if the user wishes to apply changes.

Table 7-1. @ED Control Statement, Options (continued)

Option Character	Name-1	Name-2	Description
U	Input (and Output if no Name-2)	Output	Update the symbolic element by applying corrections and create a new symbolic element cycle. For SDF-formatted files the original images will be replaced with the updated ones. The U option functions exactly as it does for SIR$ usage. In particular, an output element name may be specified which need not be the same as the input element; the output element will have the same cycle information as would the updated input element if there were no second element specified. The T option letter is reserved for internal use to facilitate this feature. In the case that the current cycle is 62 and the number of cycles in existence is less than the maximum permitted for the element being updated, the output cycle number will be the smaller of n and m-1, where n is the number of cycles in existence and m is the maximum number of cycles permitted.
X	Input	Output	Take an ERR$ exit (see Volume 2-4.3.2.2) upon a fatal or nonfatal error (batch mode only).

The ED processor operates in two modes: input and edit. In input mode, all lines entered are directly inserted into the text. In edit mode, various commands may be used to modify existing text. Changing between modes is accomplished by entering a blank line. Most editing commands implicitly reference a particular part of the text. This is accomplished by an internal cursor maintained by the ED processor. This cursor may be positioned directly by some commands (number, +number, -number) and indirectly by others (LOCATE,FIND,CHANGE).

If the P or Q options are not specified, the output file will be the same character set as the input file. If the input file is mixed, the type of the output file will be the same as the type of the label image of the input file. This means that print files will almost always be Fieldata, so the Q option will be required for ASCII editing. If a new element is being created by the explicit or implicit use of the I option, the mode will be Fieldata unless the Q option is specified.

The editor will allow editing of print files without destroying line spacing. Any new lines which are inserted have a default spacing of one. An edited file may be @SYMed as a normal print file after any editing is performed on it.

7.3. EDIT MODE COMMANDS

Table 7-2 lists alphabetically the commands available to the ED processor while in edit mode (permits manipulation of images in an element or file). Some of these commands may be abbreviated to one or two characters as specified. All may be abbreviated to three characters. All commands should start in column 1 of the input line.

When using CHANGE, INSERT, RETYPE, and the corresponding abbreviated commands, only one blank should be left between the command and the parameter image. In all other commands of more than one parameter, at least one blank must be left between each parameter. When errors are detected in a command, the command is not executed (some commands may be partially executed), an error message is given, and the next command in the runstream is executed, except in the case of the X option in batch mode.

Table 7-2. ED Processor Commands

Command	Description
ADD name ADD name num1 num2 ADD+ name ADD+ name num1 num2	This command is used to add all or portions of a file to the current file. The first form adds the whole file, and the second form adds lines 'num1' through 'num2' to the current file. The lines to be added are inserted at the end of the file unless a + immediately follows the command in which case the lines are inserted following the current position within the edit file. The 'name' is the element or filename (see Volume 2-2.6). If num2 is omitted, the single line at line num1 is added.
APPEND	Go to the end of the element or file and enter input mode thereby allowing new images to be inserted. This command may be abbreviated to A.
ASCII keyword	This command is used to specify that the character set of the output file or element should be ASCII (if keyword = 'ON') or Fieldata (if keyword = 'OFF'). If this command is not used, the character set of the output is determined from the P and Q options or the character set of the input.
AUTO num1 AUTO AUTO*	This command specifies that an automatic save of the current file is to be performed as protection against processor or system failure. The "num1" specifies that the auto save is to be performed for every "num1" input transactions which alter the file being edited. When the auto save occurs, the ED processor will type "AUTO". Entering "AUTO" with no parameter will perform an immediate auto save and reset the input transaction counter; it will also set the auto mode on, with a frequency (of input transactions which alter the file) of 131070. The effect of "AUTO*" will be the same as "AUTO", except the frequency will be left at 0 (that is, "AUTO*" is the same as "AUTO" followed by "AUTO 0"). An auto save is always performed when the ED processor goes to the top of the file, and in this case, "AUTO" will be printed if the frequency is nonzero. "AUTO 0" will terminate auto mode. Note that even if AUTO was never used, an auto recovery may be possible because of the implicit auto save each time the top of the file is reached. (Some commands, such as MOVE, may pass the top of the file more than once and will thus print "AUTO" more than once.) To recover the contents of the auto save file after a run has terminated abnormally (i.e., by system crash, terminal timeout, loss of carrier, etc.), the ED processor should be called with the A option set. If the A option is omitted and an auto file exists, the user will be asked, "DO YOU WANT AUTO RECOVERY?" An answer beginning with the letter Y will be assumed to be YES, and the effect will be the same as if the A option had been used. If an ED command, data, or the word "NO" is entered, the auto file information will be overwritten and lost. The name of the file and element (or just file) being edited will be printed before the "DO YOU WANT AUTO RECOVERY?" question is asked, to aid the user in deciding how to answer the question. If auto recovery is selected, the tables of the ED processor will be adjusted so that the output file and element will have the name of the file and element in the auto file. The tab character and tab stops will be set to the standard for the element subtype (if any) or to the default standards. Under some circumstances, the file which was in use at the time of the auto save cannot be retrieved or cannot be restored to the same status. This is the case if the file has read or write keys and the file is not assigned at the time auto recovery is attempted; it will be impossible to exit normally from the edit, and the user

Table 7-2. ED Processor Commands (continued)

Command	Description
	should free and reassign the file with the proper keys, followed by a second attempt at auto recovery. (Keys are not saved for reasons of security.) In either of these cases, the user will be informed by a diagnostic message. In some of these cases, such as when the R option was used, recovery will be impossible; the usual indication of this will be an I/O error status 5 and an empty file. The A option can be used to initiate an auto recovery at any time, not just after an abnormal termination of a run. This may be useful in other cases of abnormal functioning, such as overflow of a temporary file (which cannot be expanded) or the simple error of entering OMIT instead of EXIT. NOTE: This command should be used sparingly as it involves extra I/O and computation. Some sites may choose to set a minimum for the line count(larger than the standard of five) for this reason. An AUTO 0 terminates the auto save mode. Entering the AUTO command with no operand causes an immediate auto to be performed without affecting the auto counter.
CASE UPPER CASE U CASE NORMAL CASE N	CASE UPPER causes all input lines to be translated to upper case. In CASE NORMAL mode no translation takes place. CASE UPPER is assumed for Fieldata files or elements, and when the I option is specified without the Q option (see SHCHAR).
CCHAR char	This command sets the continuation character. When an input line to the editor has this character in it, the editor assumes that the next line or input is a continuation of this current line. This next line will be solicited in the normal manner except that a + will precede the solicitation. The character is initially set to a character which cannot be typed in. The character can be reset to this non-enterable character by using this command with no 'char'.
CHANGE /string-1/string-2/m G C /string-1/string-2/m G CHANGE /string-1/string-2/> m G C C?	This command searches a specified number of text lines for (as with LOCATE command except that CHANGE starts with the current line and LOCATE starts with the following line). When and if the desired string, 'string1', is found, 'string 2' is substituted for it. The number of lines to be scanned is indicated by 'm'. The global indicator, 'G' tells whether to change all occurrences of 'string1' in the range of lines or just the first occurrence in each line. A 'G' means all occurrences and no character means just the first. The '/' may be any character which does not occur in 'string1' or 'string2' except a blank. The 'm' may be omitted in which case 1 is assumed. Instead of using 'm' and 'G', a user may change all subsequent occurrences in the file by using the word 'ALL' (which may be abbreviated A) where 'm' is usually specified. Instead of using a large value for m, the user may specify the word REP (which may be abbreviated R) to indicate that the first occurence on each line of the rest of the file is to be changed. A special option allows the erasure of all characters following a given string in a text line. This is accomplished by placing a > immediately following the last string delimiter in the CHANGE image. If all specifications are omitted, the last CHANGE command will be executed again. The contents of the last CHANGE command can be printed by "C?".

Table 7-2. ED Processor Commands (continued)

Command	Description
COMP e COMP* e COMP! e COMP:v e	This command computes the value of an integer expression. If the command is entered with a trailing asterisk, the computed value will be printed in decimal; if the command has a trailing exclamation point, the value will be printed in octal; otherwise, nothing will be printed. The computed value is always retained for later use with either the LPSUB or the LPTST command "V" specification. If the command is followed by a colon, the colon must be followed by a variable v, which may be any of the 27 possibilities X, XA, XB,..., XZ. The value computed by the expression will be saved as the value of the designated variable as well as for the V specification. The expression e may use the operators +, –, *, and /, as well as parentheses for grouping. The only operands allowed are integers, the variables (X, XA, etc.), and the specifications I, L, N, V, LC, CC, and LG, which have the same meanings as they do for the LPSUB command. Integers used in the expression are interpreted as octal (base eight) if they have a leading zero (standard 1100 Series convention); otherwise, they are assumed to be decimal (base ten). For example, the numbers 015 and 13 represent the same value.
CPT	This command prints out the SUPs used so far in the present run. The form of the message is (hours)H (minutes)M (seconds)S. The seconds field is given to four decimal places.
CPUNCH num1 num2 device CPUNCH num1 device CPUNCH device CPUNCH	This command is used to punch parts or all of a file at an onsite card punch. The syntax has the same meaning as with then SITE command. After the command is entered, a message MSG? will be typed out. The line typed in will be sent to the system console before the cards are punched. A "C" option is assumed for the @SYM command by CPUNCH so that a remote site may be designated for the output as well as an onsite printer. If no device is specified, a default of "CP" is used. If the @SYM operation requested by the (LN)SITE or CPUNCH command fails due to an invalid symbiont name, the user will be given one additional chance to enter a valid name. For a second invalid name or for any other error, the ED processor will print the name of the file for which the error occurred, thereby making it possible for the user to select an appropriate disposition of the file. At the time the error message is issued, the file created has been freed and is catalogued. Therefore, the user should either SYM it for printing/punching using a valid site-id or symbiont name or else delete it from the directory.
CSF Executive control statement	This command is used to submit a control statement via CSF$ (see Volume 2-4.10.1.1). Only statements valid for CSF$ may be submitted. The control statement must start in column 5.
DELETE num1 num2 D num1 num2 DELETE num1 D num1 DELETE+ etc. D+ etc.	This command is used to delete lines from the text.The first form deletes lines 'num1' through 'num2'. The second form deletes the next 'num1' lines starting with the current one. A '+' following the command name will cause the editor to be positioned after the lines deleted; this saves one entire pass over the file.

Table 7-2. ED Processor Commands (continued)

Command	Description
DITTO num1 DITTO num1 num2	This command allows duplication of other lines in the file. The duplicated lines are inserted at the present position in the file. The first form results in the one line at 'num1' being inserted in the present position. The second form results in all lines 'num1' through 'num2' being duplicated at the present position. Care must be exercised to be sure the most current line numbers are used. At the completion of the DITTO, the pointer is positioned at the last line inserted; this saves one entire pass over the file.
DOC lines column P	This command is used to add documentation to existing images. 'Lines' is an integer indicating the number of lines to be documented starting at the current line. 'column' indicates the column at which the comment is to be inserted. P, if specified, indicates that '. ' (period-space) is to be automatically inserted before the comment. When this command is entered, each image will be printed out to the proper column via ATREAD$. The user may then enter his comment or one of the following: @EOF – no comment for this line. @EOF n – the line will be retyped to the specified column plus n is one through nine. This is used for lines of code which extend beyond the normal comment column. @EOF x – discontinue documentation. After completion of this command, the editor is positioned following the last line read by the command.
EXCH char octal number	This command is used to allow input of characters not represented in the keyboard character set. 'char' is the character which is to be used to stand for the number whose internal ASCII representation is 'octal number'. When 'char' occurs any place in an input line it will be replaced by this character. An EXCH with no parameters disables this feature. As an alternative to the 'octal number' for ASCII control characters, the character name (e.g., NUL, BEL, HT, etc.) may be used.
EXIT	This is the command used to take a normal exit from the ED processor. All the corrections will be applied to the designated file and a normal exit will be taken. When an overflow occurs on EXIT, the editor will automatically expand a cataloged file until it is large enough to hold the output; for temporary files, a message is generated, and the user may recover editing using the A option.
FC mask FC*n mask FC,n mask	The FC command behaves in the same way as the FIND command except that all occurrences are flagged in the remainder of the file. By immediately appending an asterisk followed by a number onto the command word, the search will stop after that number of occurrences of the target are found. (See 7.6.5.4.) If 'mask' is omitted, the mask from the last F/FC is used. A comma followed by a number indicates the number of lines to search.
FIND mask F mask F,n mask F?	FIND searches for an image which corresponds exactly column for column starting at column 1 with the 'mask'. Transparent characters may be used in the mask which will test successfully with any character in the column. The normal transparent character is a blank, but an alternate may be designated with the TCHAR command. The search begins with the line following the

Table 7-2. ED Processor Commands (continued)

Command	Description
	current one and proceeds until a match or end-of-file is detected. The command may be abbreviated to F. By immediately appending a comma followed by a number onto the command word, the search will be limited to that number of lines. To allow searching only the current line in LOOP mode, a test can be made for FIND or NOFIND condition. This option is invoked by following the command name immediately by a period. Note that without this option, FIND begins with the following line and not the current line and repeats. If 'mask' is omitted, the mask from the last F/FC command is used. "F?" will print the default mask. Both F and FC recognize the TAB character. When a TAB character is encountered, all characters are assumed to match up to the next tab stop, and matching resumes at that point. This can save much typing and counting.
	This command position the editor at a line (specified by number) in the input text, even after the input file has been edited with insertions, deletions and other modifications. If the desired line has been deleted or altered in some way, the message: REQUESTED LINE DELETED OR ALTERED will be printed, and the editor will be positioned at the next existing line. This command is useful when a user is editing text while working from a listing. This command cannot be used with print files.
	This command behaves exactly the same as the INSERT command except that the line is inserted before instead of after the current line.
number term-sub number term-sub	This command allows inline editing of a given line. If 'number' is blank, the current line is assumed to be the one to be edited, unless the command is followed by a +, in which case the next line will be the one to be edited. Otherwise the editor proceeds to line 'number'. The line will be printed out. The user can then enter editing information directly below the line to modify it. The "term-sub" field is optional; the normal termination character for editing commands is the exclamation point ("!"), but this field can be used to specify a different character if the information to be edited contains exclamation points. (Note that if the "number" field is to be omitted, the "term-sub" field must not be interpretable as an expression.) The editing characters to be used are: I – The string following this command is inserted following the character immediately above the I. The string is delimited on the right by the termination character '!'. R – The characters following the R will replace the characters immediately above them. A ! is required to terminate replacement. D – The characters in the line above are deleted between D and the !.

Table 7-2. ED Processor Commands (continued)

Command	Description
	More than one of the insert, delete, and replace operations may be requested in a single INLINE edit. The letters I, R, and D may be entered in either upper- or lower-case. Before using this command in @@CQUE mode on a TTY-like terminal, the user is advised to enter the command "OFF U"; otherwise, the alignment of the editing line will be incorrect. The alternate character specified by "term-sub" remains in effect for only a single command. Because of alignment problems, the entry of the editing commands following a MCCHAR character should be done with caution.
INPUT	This command directs the editor to enter input mode. In this mode everything which is typed in is inserted in the file until an exit from the mode is taken. This is especially useful when large volumes of input are to be entered. Exiting from this mode is accomplished by typing an @EOF when in EOF mode (see ON and OFF commands), a carriage return, or blank line when not in EOF mode, or by @EDIT with no command. Tabs are recognized in this mode.
INSERT string I string	This command is used to insert a line following the line presently pointed at by the editor. The new line will then be the point at which the editor is positioned. The string to be inserted starts after the first blank following INSERT. If a '+' immediately follows the command, the string may be input on the next line (this provides more room, as a full input solicitation will not occur, and the command itself is not present. If the command is entered without a "string" when not in EOF mode (see 'ON' command) the editor will switch to input mode. In EOF mode this simply results in the insertion of a blank line.
LAST	This command directs the editor to move to the last line in the file and stay in edit mode. The last line cannot be altered after this command has been used until the file position is changed. There are six commands which are illegal after the LAST command has been used (until a line has been added or the file position is changed); these are CHANGE, DELETE, DOC, IB, INLINE, and RETYPE. An attempt to use any of these immediately after entering the LAST command will produce a diagnostic (and error termination in batch mode if the X option was set). In most cases, any command (such as GO or entering a number) which attempts to move to the current line number will simply cause the current line to be typed out. Because of the special situation which exists after the LAST command has been used, the ED processor will actually do the move if a transfer to the current line number is attempted after a LAST. This will permit the above six commands to be performed.

Table 7-2. ED Processor Commands (continued)

Command	Description
LC string LC quote-char string quote-char LC*n string LC,n string	LC behaves as LOCATE except that all occurrences of the string in the remaining text are located. Just before each line containing an occurrence is typed out, the line number is typed out. By immediately appending an asterisk followed by a number onto the command word, the search will stop after that number of occurences of the target are found. A comma followed by a number indicates the number of lines to search. If 'string' is omitted, the string from the last L/LC command is used.
LCHAR char	This command sets the quote character for the LOCATE command. The default character is quote ('). A non-input character will be assumed if 'char' is a blank. By immediately appending an asterisk followed by a number onto the command word, the search will stop after that number of occurrences of the target are found.
LIMIT keyword num1 num2	This command allows setting of left and right column limits for CHANGE, LOCATE, and PRINT. keyword is CHANGE, LOCATE, or PRINT (each of which may be abbreviated to its first letter). num1 and num 2 represent the left and right column limits. If only one column number is specified, it designates the right limit, with one assumed as the left limit. If no column numbers are specified, the column limits are set to the default values (1,132). If keyword is CHANGE, then this command sets limits on the columns which will be searched by the CHANGE command. This is useful for protecting areas of text lines in a file. If keyword is LOCATE, then this command sets limits on the columns which will be searched by the LOCATE and LC commands. If keyword is PRINT, then this command sets limits on the columns which will be printed by the output commands (PRINT, LNPRINT, QUICK, LNQUICK, PUNCH, CPUNCH, SITE and SITE and LNSITE) as well as by other commands which print lines of text. As before (using the LIMIT command), print column limits specified by the user are rounded to the nearest ASCII word boundary, e.g., LIMIT PRINT 8 9 will cause columns 5 to 12 (words 2 and 3) to be printed by the PRINT command. Limits specified by this command may be overridden on any single command by the use of immediate column limits specifications (see 7.6.5).
LOCATE string LOCATE quote-char string quote-char LOCATE,n string LOCATE. string L string	This command is used to search the text for a given string of characters. The search begins at the line following the current line and proceeds sequentially through the text until a find is made or the end of file is encountered. The first form ignores multiple blanks in the images. The second form requires that the text image be exactly the same as the string within the two quote characters. This command may be abbreviated to L. By immediately appending a comma followed by a number onto the command word, the search will be limited to that number of lines. To allow searching the current line only in LOOP mode, a test can be made for FIND or NOFIND condition. This option is invoked by following the command name immediately by a period. Note that without this option, LOCATE begins with the following line and not the current line. If 'string' is omitted, the string from the last L/LC command is used. If the LC variable is to be referenced after a LOCATE, the quoted form should be used; otherwise, the position indicated may include leading blanks before the string located.

Table 7-2. ED Processor Commands (continued)

Command	Description
LOOP n start increment LOOP n LOOP? LOOP+ LOOP* LOOPI n start increment	This command allows repetitive execution of a group of statements, n is the number of times the statements are executed (the default value is 0), start specifies the value of the counter (the default value is 1), increment indicates that the current line number is incremented by this amount on each iteration (the default value is 1). The commands to be executed will then be solicited with an LP**. (See LOOP Operations 7.4.) The following commands are often used with the LOOP command: LPSUB (See 7.6.2) LPTST,n, (See 7.4.3) XTI (See 7.4.4) LPEND (See 7.4.5) LPJUMP (See LPJUMP) LPX (See 7.4.6)
LPJUMP x	This command is used to perform an unconditional transfer of control to a labeled point in a loop, a macro, or in the input stream. The "x" denotes a label; a label may contain any ASCII characters except blank or comma, but lower case characters are treated as equal to the corresponding upper case characters. Transfers within a loop or a macro may be backwards or forwards, but transfers within the input stream may only be forwards. The specified transfer point is a line whose format is one of the following: :x @EDIT :x where the colon must appear in column 1 (column 7) and must be immediately followed by the label with no intervening blanks. Such a labeled line may not contain any other editor commands, and has no effect if executed rather than used as a LPJUMP target. The search for a label within a macro or loop is downward to the bottom of the outermost loop and then downward from the top of the currently active loop. If the label cannot be found within the loop, an error message is printed and the loop is terminated. Labels should be unique within any individual macro and within a loop entered from the runstream; this restriction is not absolute, but the search algorithm should be understood thoroughly before using non-unique labels. It is inadvisable to transfer out of the currently active loop. The label on the LPJUMP command may be generated by LPSUB substitution, but it is not permitted to generate any part of a label line itself by use of LPSUB. When LPJUMP is used in the input stream, the name of the jump target will be printed, and input solicitation will indicate that a jump is in progress. The "@@X C" command may be used to stop a jump as for stopping a LPTST skip.
MACRO n MACRO* n MACRO? n, n, n, ..., MACRO? MACRO??	For each of the first three forms, "n" denotes the name of a macro. Macro names are unique by the first three characters only. The first character must be alphabetic, but the other characters may be alphanumeric; the dollar sign may also be used. If a macro name duplicates the name of an existing ED command, the definition will be accepted, but the macro will never be callable. The first form specifies the entry of a macro definition. Macro definitions are entered exactly like LOOP definitions (including the use of @EOF and @EOF L),

Table 7-2. ED Processor Commands (continued)

Command	Description
	except that they are solicited by the typeout "MAC*". A macro definition operation will destroy any stored loop, so that "LOOP!" will be invalid. The second form specifies the deletion of the definition of the specified macro. It is not necessary to delete a definition before redefining a macro; this is done automatically. The third form will list the text of one or more defined macros, the fourth form will list the names of all defined macros, and the fifth form will list the text of all defined macros. (See 7.5.) (For purposes of listing macros, the set of defined macros contains only those known to the ED processor; those contained in program files as elements named "mac-ED-MACRO" but which have not been called will not be listed.)
MAIL user-id	This command allows messages to be sent and received in communication with other users. The editor will then solicit 100 lines of input with: MAIL** If the desired message is to be less than 100 lines the mode can be terminated by entering an @EOF. After the message is received by the designated person it will be deleted. The ED processor will never search for mail in batch mode; therefore, there will never be a solicitation "DO YOU WANT YOUR MAIL?". In demand mode, the ED processor will search for mail on entry (after the sign-on line) in either edit or input mode. If the user's response to the solicitation "DO YOU WANT YOUR MAIL?" comes from an add file, does not begin with "Y", and is not "NO", then it will be treated as a command (if edit mode) or the first input line (if input mode). NOTE: The ED processor will look for mail only the first time it is called in a run, rather than each time the ED processor is invoked. The LOOK mode has been deleted. The system generation parameter MAILINIT controls whether the ED processor will look for mail each time it is called: As released, the value is 0; setting it to 1 will restore the previous mode of operation. A system generation parameter MAXMAIL is provided to control the size of mail files; to encourage larger (and thus fewer) files, the ED processor is released with this parameter set to 100 instead of the previous value of 10.
MAXLINE number	This sets the maximum length (1 - 132) to which a line may increase. If it is exceeded, the line will be truncated. The default is 132. The largest acceptable line length is configurable in the ED processor but also depends on what is permitted by the Operating System.

Table 7-2. ED Processor Commands (continued)

Command	Description
MCCHAR char	This command specifies a character which is used to separate multiple commands given on a single line. Each occurrence of the multiple command separator character terminates a command or input line and begins a new one. For example, the user may enter: MCC # GI 453#C /ABC/XYZ/ to make changes on input line 453. If the last non-blank character on a line is the MCCHAR character, the effect is that of a blank line following the last command on a line. When using the LPTST command with this feature, the skip count is decremented by 1 for each line (NOT command) scanned, including the remainder of the line containing the LPSUB command, if there were additional commands on that line. The LPSUB command substitution count applies to commands rather than lines. Care is required, however, if the substituent contains the MCCHAR character. In order to be recognized, labels (":xxxx") must appear as the first command on a line. This feature is normally disabled, and it may be discontinued by entering MCCHAR with no character present. This character is also recognized in input mode and allows entry of several text lines in one line of input. In input mode, each command entered using the @EDIT feature must begin with @EDIT if more than one command appears on a single line through use of the MCCHAR character. Since the @EOF image is a system image rather than an ED image, it must always appear by itself on a single line.
MOVE num1 MOVE num1 num2	This command performs the same operation as the DITTO command except the original lines are deleted after the duplication has taken place. The syntax is the same as for the DITTO command. Care must be exercised to be sure the most current line numbers are used. At the completion of the MOVE, the pointer is positioned at the original line number.
MSCHAR char	This command sets a character which will be translated to a master space (more commonly known as the "at" symbol, @) when it is input in column one of input lines in input mode. If 'char' is a blank, no master space translation is available.
number +number -number	These commands are used to position the editor at a desired line in the text. The first form directs the editor to line 'number'. The second form directs the editor to move to the position current line plus number. The third form directs the editor to move to the position current line minus 'number'. When the specified line is located, it is typed out (if not in BRIEF mode), and modifications may be made to it. If it is desired to insert lines before line 1, 0 may be typed in. This will position the editor immediately before the first line.

Table 7-2. ED Processor Commands (continued)

Command	Description
OMIT	This is the command to be used if the user does not want his corrections to be applied to the file on exit. The input file will remain as it was at the beginning of the editing session, and the output file, if any, will not be produced. In read-only mode, EXIT is synonymous with OMIT.
ON special mode ,..., special mode OFF special mode ,..., special mode	This command is used to define some special modes within the editor. ON turns the mode on, and OFF turns it off. The special modes are: BRIEF – do not echo corrected images for CHANGE and DITTO. DSPLIT – delete lines transferred by SPLIT command. EOF – special mode where blank lines may be entered. INP command enters input mode and @EOF exits from input mode to edit mode. While in input mode blank lines may be entered. Also the INSERT command with no image following will enter a blank line. INPSEQ – When on, input solicitation (if active) will include the input line number in parentheses. (This is the line number referenced by the GI command.) LSTINP – When on, input lines and commands to the ED processor will be printed in the run listing. When off, input will not be listed. This mode is also controlled by the L option on the ED processor call statement. This mode may be turned on to trace the statements executed by a LOOP or MACRO call. MEMORY – remember modes on successive executions. NUMBER – precede each line printed out with its number. PCNTRL – Print control images will be printed if this mode is on. QUICK – compress extra blanks out of all output to device TRDINP – When on, demand input is via TREAD$. When off, demand input is via READ$. There is no effect in batch mode. This may be turned off in @@CQUE mode to permit the fastest possible typing speed. UNISCP – allow correct character placement on UNISCOPE terminal with the INLINE command. XBRIEF – do not echo lines transferred by SPLIT or ADD. All of the modes may be abbreviated to one letter.
OPR string OPR*string	This command is used to send a message to the system console. The first form sends the message 'string'. The second form does the same, but also solicits an answer. The string may not be more than 50 characters or it will be truncated.

Table 7-2. ED Processor Commands (continued)

Command	Description
PCC n m PCC n PCC+	This command behaves like the PRINT command, including the use of the form PCC+. Any ASCII control characters on a line will be mapped into the character whose code is 0100 larger, and any lower case characters on a line will be mapped into the character whose code is 040 smaller. The line will then be printed, followed by a second line which will contain spaces below any characters which were in the 64-character ASCII set, "L" below any characters which were lower-case, and "C" below any characters which were control characters. Thus, if a line consists of the characters upper-case a, lower-case n, and BEL, PCC would print that line as: ANG LC
PRINT num1 num2 PRINT num1 PRINT	This command is used to print out lines of text.The first form prints lines 'num1' through 'num2'. The second form prints the next 'num1' lines. If the command is immediately followed with a + the printing starts with the next line instead of the current one (example: PRINT+ 3). The third form prints the entire file from the top. If no number or recognizable symbol follows the command, a 1 is assumed; that is, the present line will be printed out. This command may be abbreviated to P. See also PCC command.
PUNCH num1 num2 PUNCH num1 PUNCH	This command is used to punch paper tape for form II paper tape input (see Volume 2-8.2.1.2.2) at a terminal which has punch and read hardware. The syntax for this command is the same as that for the PRINT command. When the command is entered, the following response will be given: DEPRESS PUNCH ON The processor will then pause to allow the user to push the punch on button on the paper tape punch hardware. After pausing, the designated lines will be typed out which will cause the paper tape to be punched at the same time. Rubouts will be punched at the start and end of the tape. The tape so produced can be used as normal form II input.
QUICK num1 num2 QUICK num1 QUICK	This command prints lines with all nonsignificant blanks omitted. This provides a fast method of examining areas of the file. 'num1' and 'num2' are the same as on the PRINT command. Plus (+) may also be used on the second form with the same meaning. This command may be abbreviated with a Q.
REMARK text REMARK*text REMARKtext REMARK?text	This command is intended primarily to incude commentary in loops, although it may also be useful in an @ADD file. The "text" may consist of any comment the ED user may wish to include. If the form REMARK* is used, the text will be printed, prefixed by "REM*:", which may be useful in conjunction with the XTI command. The form REMARK has the same effect as REMARK*, except that the prefix will not be printed. The form REMARK? is used to solicit a response; this response may be accessed later via the LPSUB and LPTST QU specification. The LPSUB command can make substitutions into the text field of a REMARK command.

Table 7-2. ED Processor Commands (continued)

Command	Description
RETYPE string R string	This command is used to completely replace the current line with the string following the first blank after the command. A + may be used after the command with the same meaning as with the INSERT command.
RP number	This command is used to set a repeat counter for the INSERT command. Any insertion will be repeated 'number' times. The counter remains in effect until explicity reset to 1.
SCALE SCALE n1 SCALE n1 n2 SCALE n1 n2 n3 SCALE! etc.	This command causes a line to be printed which can be used as a measuring scale for column sensitive operations. It consists of the digits 0 through 9 repeated with the digit D falling in a column whose number is congruent to D modulo 10. If n2 is omitted, 72 is assumed; if n1 is omitted, 1 is assumed. If the command is entered in the form "SCALE!", a second line will be printed, consisting of the tens (and possibly hundreds) digits. If both n1 and n2 are present, there may be a third parameter, n3, which indicates an offset value to be added to the digit printed in each column. This may be of use in lining up columns for input with the "I" command, as by entering "SCALE 8 80 3", where the actual values used will depend on how many digits are printed by input solicitation. If n3 is omitted, a value of zero is assumed.
SHCHAR char	This command specifies a character for case shifting for use with devices (teletypewriters, card readers) which do not have lower-case capabilities. When the specified character is encountered on any line processed by the ED processor, all following upper-case alphabetic characters will be translated to the corresponding lower-case characters until another shift character is encountered. The user should always specify "CASE NORMAL" before enabling this feature. If no character is specified, the shift feature is disabled, and shift mode is turned off. The printout from the STATUS command will indicate whether shifting is active or not. The shift character itself will be deleted from any lines in which it appears.
SEQ.id i,j SEQ.id col i,j	This command causes sequence numbering to be inserted into a specified set of columns on each image in the file. The value of i is the starting number, and j is the increment. Omitted values are given a default of 100; if i is omitted, i will be ignored. The id field may contain any ASCII alphanumeric characters. If it is omitted, the sequence field will contain only the sequence number. Sequence numbers are printed with leading zeros. If the sequence number is or becomes too large to fit in the field, it will be reduced modulo the appropriate power of ten. If the id field is the same size as the sequence field, no sequence numbers will appear, just the id. If the id field is larger than the sequence field, an error will result. The col specification defines the column limits of the sequence field in the format for immediate column limits specifications as on the CHANGE command ([m,n], etc.). If the col specification is omitted, columns 73 through 80 are assumed. The use of a column specification may be particularly helpful when editing COBOL or BASIC elements, to create required sequencing or line numbering.

Table 7-2. ED Processor Commands (continued)

Command	Description
SET tab1 tab2 tab3 ... tabn SET+ tab1 tab2 tab3 ... tabn	This command is used to set the tabs for the commands which allow them as explained above. As many tabs as desired may be designated. Each SET command redefines all previous tabs, and so a SET with no tabs clears the tabs. If no SET has been performed, the tabs are set as follows: 5,7,73 if the input element is type FORTRAN or FORTRAN PROC, 8,12,73 if the input element type is COBOL or COBOL PROC, 6,11,16,21,26 if the input element is type ALGOL, PLUS or PL/I, 11,21,39,73 otherwise. If the command is entered as 'SET+', the indicated tabs will be added to the set of tabs already in use. In this case, the first tab specified must be larger than the largest existing tab. This feature can be used in a loop to set up a number of evenly spaced tab stops as follows: SET LOOP 15 6 5 LPSUB L,$ SET+ $ @EOF The first SET is necessary to clear any existing stops. The loop given will set tab stops at columns 6, 11, 16, 21, ... ,76.
SITE num1 num2 device SITE num1 device SITE device	This command is used to direct output to an onsite printer. The meanings for 'num1' and 'num2' are the same as for PRINT except that if no numbers are given, the third form is assumed 'device' specifies the symbiont name to which output is sent. If the field is not specified, PR is assumed. After this command is entered, a message as follows will be typed out: HDG? The line typed in here will be used to head the onsite output. Periods must not be used in this header as anything beyond the period will not be printed. After the output is done, the following will be typed: MSG? The user should enter here the information necessary to indicate where and to whom the output should be returned.
SPLIT name SPLIT name num1 num2	This command is used to build new elements or files from portions of a current file. Note that the default for the ON command DSPLIT is on and data lines tranferred by SPLIT will be deleted. The first form caused all lines preceding the line currently pointed at to be reproduced as the designated file. The second form causes lines 'num1' through 'num2' to be reproduced. An 'I'

Table 7-2. ED Processor Commands (continued)

Command	Description
	immediately after the SPLIT command causes the whole file to be copied. The character set (ASCII or Fieldata) of the new file is the same as the character set of the output file. When used in read-only mode, the SPLIT command will always function as if DSPLIT mode is off. If DSPLIT is on, a diagnostic message will be printed.
SSP n SSP?	The SSP command has two formats. The form "SSP?" prints out the line spacing value for the current line. The form "SSP n" sets the line spacing value for the current line to n; if n is omitted, a value of 1 is used. This command may be used to override the automatic value of 1 for line spacing for newly inserted lines. This command may only be used when editing print files. In read-only mode, only the "SSP?" form is valid.
STATUS special mode ,..., special mode STA*	This command is used to request the status of special modes set by the ON and OFF commands. If no special modes are specified, the status of all will be listed, along with other status information. The form "STATUS*" with no specifications will not list the mode values, but will give the other status information.
STK x	The STK command provides a way of remembering the set of ON/OFF modes and is intended primarily for use in macros. The stack depth limit is five. If x is omitted or is UP, the modes will be saved. If x is DN or DOWN, the modes will be restored. Other specifications are erroneous. No checks are made for the number of "UP" operations exceeding five or the number of "DN" operations exceeding the number of "UP" operations.
TAB tab-char TAB	This command is used to specify which character is to be used as a tabulator character. This character is recognized on the INSERT, IB, and RETYPE strings and is recognized on all input when in the input mode. The character is not transmitted to the file and behaves just as a tab on a typewriter. If no character has been specified, a semicolon (;) is the tab character. If no TAB has been performed, the tab character is set as follows: ' # ' (number sign) if the input element is type ALGOL, BASIC, PLUS or PL1. ' - ' (underline) if the input element is type DOC. ' ; ' (semicolon) default. A blank may also be used as a tab character, by specifying the tab character as "SP". ASCII control characters may be specified by name (NUL, SOH, etc.) as well as by direct entry. Entering this command with no character specified disables the tab character.
TCCHAR char	This command allows setting a transparent character for the CHANGE and LOCATE/LC commands. If specified in string 1 of the CHANGE command or string of LOCATE/LC, the character char will match any character in the text. There may be any number of occurrences of char in string 1 of a CHANGE command. If the transparent

Table 7-2. ED Processor Commands (continued)

Command	Description
	character occurs in string 2 of the CHANGE command, each occurrence of the character will stand for the character in the original image matched by the corresponding occurrence of the transparent character in string 1. If the number of transparent characters in string 2 exceeds the number in string 1, the excess transparent characters in string 2 will stand for themselves. For example, the command sequence: TCCHAR % C /%/% / G will place a space after each character of the current line. No char field disables the feature. TCCHAR SP sets the transparent change character to blank. The feature is initially disabled. The TCCHAR will also be recognized as a "match anything" character for the LOCATE and LC commands.
TIME	This command prints out the date, time and cycle information and the name and type of the output element or file.
TYPE processor-mnemonic TYPE* processor-mnemonic	Sets the processor type for symbolic element output. The processor mnemonics are: ALG, APL, ASM, ASMP, BAS, COB, COBP, DOC, ELT, FOR, FORP, FLT, LSP, MAP, PLS, PL1, PNC, SSG, SEC, TCL. Octal numbers may also be used instead of the mnemonic. If the TYPE command is entered as "TYPE* xxx", then the tab character and tab settings appropriate to the type xxx will be established, replacing any tab character and settings currently in effect.
UNDO	This command causes all changes made since the last return to the top of the file to be ignored, and the editor is positioned at the top of the file. If no changes have been made, a diagnostic message is printed, and the file position remains unchanged. Since some commands (for example, DELETE, MOVE, and DITTO) implicitly go to the top of the file, it will not be possible to recover from their effects in all cases. Caution should be exercised when using the UNDO command.
UP	This command is used to cause the editor to behave as if the U option had been specified on the control statement. This is used if the entry to the editor was made with an R option.

Table 7-2. ED Processor Commands (continued)

Command	Description
WAIT n	This command allows invocation of a voluntary wait. If @@X C is used to interrupt the waiting, the interrupt will occur after at most a 30 second delay. The time may be specified as a decimal number of seconds or by parameters of the form nH, nM, and nS, where any combination is permitted, n is a decimal integer of six digits or fewer, and values of n need not be less than 60. The keyword UNTIL specifies a time of day to be waited for, and AM or PM may be used, as well the 24 hour clock.
XPC print control image	This command is used to submit a print control image via APRTCN$. (see Volume 2-5.4.2 for description.) The print control image must start in column 5 and may contain one or more print control functions, limited only by the length of the line. See Table 5-2 of Volume 2 for the list of valid print control functions.

7.4. LOOP OPERATIONS

7.4.1. LOOP Command

Purpose:

Allows repetitive execution of a group of statements.

Format:

```
LOOP n start increment
LOOP! n start increment
LOOP?
LOOP n
LOOP+
LOOP*
```

Parameter:

n	Specifies the number of times commands are executed (default value is 0).
start	Specifies the initial value of the counter (default value is 1).
increment	Indicates that the current line number is to be compared if it is N (default value is 1).

Description:

When the desired number of commands has been entered, the user should enter the command LOOP n where n is the number of times the commands are executed. The commands to be executed will then be solicited with an LP**.

The LOOP command is ended with an @EOF (with a nested Loop, only the outer most loop is ended with @EOF; LPEND will end inner loops). If an @EOF in the loop is desired, this may be accomplished by @EOF L. The sentinel character L will allow the LOOP entry to continue. An erroneous loop may be aborted before execution begins by @@X C followed by @EOF. The execution will stop after 'n' executions, when the bottom of the file or element is reached, or if @@X C is entered.

The format "LOOP?" (no parameters) will print out the currently stored loop. The format "LOOP+" (no parameters) will expand the space available for storage of loops and macros by 512 words; this may be needed for exceptionally large loops. LOOP+ may be used more than once. The format "LOOP*" may be used to contract the space used for loop and macro storage when it is no longer needed. An attempt to contract to less than the initial size will be ignored. In order to release space, there must be no information stored in the area to be released, or else the command will be ignored. This means that macros no longer in use must be deleted with "MAC*" before attempting to release storage. LOOPI causes the currently stored loop to be executed again with new parameters. LOOP? will print out the currently stored loop.

Loops may be nested up to ten deep. All nested loops must be terminated by a corresponding LPEND command. The LOOP command is identical for nested loops, except that the exclamation point in 'LOOPI' will be ignored, as it is meaningless. Macro calls are treated as nested loops.

Note that any command which causes the editor to pass the start of the file, such as one which causes it to back up its position in the file (for example, –n, 0, or FC) will terminate a loop which did not request non-stop operation (by specifying the iteration count with a trailing exclamation point). The DELETE, MOVE, and DITTO commands are treated as special cases and will not terminate a loop. The LAST command goes to the end of file, but does not pass the start of the file, and thus will not stop a loop. The SPLIT command will stop a loop, as will unsuccessful FIND and LOCATE commands which reach the end of file.

Example:

```
0:▷LOOP 99999
LP**▷LOCATE ABC=
LP**▷i  PRINT ABC
LP**▷@EOF
```

This loop will locate all assignments to the variable ABC and insert a print statement following the assignment statement.

7.4.2. LPSUB Command

Purpose:

Allows the replacement of characters in one or more following command or data lines with variable information.

Format:

LPSUB $v_1,s_1\ v_2,s_2 \ldots v_i,s_i$

LPSUB,k $v_1,s_1\ v_2,s_2 \ldots v_i,s_i$

Parameters:

k	Number of lines (default value is 1 that is, the next line only).

v_i One or two characters indicating the value to be subsituted.

s_i One or more parameters denoting the character to be subsituted for and other specification as specified in Table 7-3.

Table 7-3. LPSUB Specifications

Spec.	Format	Description
I	I,c	Input line counter.
L	L,c	Loop counter for currently executing loop of a nest.
Ln	Ln,c	Loop counter for loop at depth n of loop nest (0 is outermost loop).
LD	LD,c	Loop nesting depth.
N	N,c	Line counter.
LC	LC,c	Column in which last string found by the LOCATE or LC command starts.
CC	CC,c	Column in which last string changed by the CHANGE command starts.
LG	LG,c	Length of current text line (note that blank lines have a length of 1).
V	V,c	Value computed by most recent COMP command.
T	T,c	Time of day.
D	D,c	Date.
T1	T1,c	Time of day (all numeric, format hhmmss).
D1	D1,c	Date (all numeric, ISO format yymmdd).
SP	SP,c	Spacing value for current line (1 for non print files).
TX	TX,c,x,y	Portion of current line of text. The character c is replaced in the following lines by the y characters beginning at character position x. If y is omitted, a value of 1 is assumed; if both x and y are omitted, the entire line is assumed. If y has an asterisk (*) suffixed, trailing blanks will be removed from the substituted string. A value of zero for x will be treated as 1.
QU	QU,c,x,y	Portion of response to most recent 'REM?' command. The x and y parameters have the same meaning as for TX.
MA	MA,c,n,x,y	Parameter submitted on most recent call to the macro specified by the name n. The meaning of x and y is the same as for TX. If x is explicitly entered as zero, the string will begin with the delimiter which followed the macro name on the macro call line. That is, the delimiter is treated as the zeroth character of the parameter. If x is omitted, it is treated as 1. The asterisk may be used following y as for the TX subsitution.
U	U,c	User-id (as determined by the ED processor).
R	R,c	Run-id (original).

Table 7-3. LPSUB Specifications (continued)

Spec.	Format	Description
RG	RG,c	Generated run-id (guaranteed unique by the EXEC).
SI	SI,c	Site-id of input device.
NN	NN,c	String whose value is null, LN, IL, or NI, depending on what prefixed the most recent uncompleted LOOP command or macro call. This allows a macro, for example, to be called as "LNxxx" and substitute the characters before printing commands in the body of the macro.

Description:

LPSUB allows the replacement of characters in one or more following command lines of a loop with variable information. Each v_i is one or two characters indicating the value to be substituted, and s_i is one or more parameters denoting the character to be substituted for and other specifications as given.

Each s_i indicates the character for which the indicated substitution is to be made, and other parameters for substring selection and macro name specification. The specification k indicates how many lines are to be affected by the LPSUB command; if k is omitted, 1 is assumed (that is, the next line only). An LPSUB overrides any previous LPSUB still in effect. All occurrences of each character specified in each si on the following n images will be replaced. An example of the use of LPSUB is as follows:

```
LOOP 6
LPSUB L,$
I ;L,S$;A0,IMAGE,X7;.          Note the semi-colon used as tab character.
@EOF
```

which will insert the six lines

```
L,S1        A0,IMAGE,X7    .
L,S2        A0,IMAGE,X7    .
etc.
```

into the file.

Only one substitution of each type may be active at a time. (Each Ln is considered different for different values of n.) This means that MA and QU substitution may be used simultaneously, but it is not possible to substitute parameters from two different macros at the same time.

LPSUB may be used outside of loops in the same way and for the same reasons as LPTST. The substitution will affect the specified number of input lines.

The LPSUB counter is *not* decremented for lines skipped by LPTST or LPJUMP. Therefore, if the number of lines to be substituted is different on different loop iterations, the command "LPSUB,0" may be necessary to turn off substitution. If this is not done, there is the possibility that an LPSUB command could modify itself on a subsequent iteration of the loop, producing unexpected and mystifying results.

7.4.3. LPTST Command

Purpose:

Allows conditional execution of statements in a loop.

Format:

LPTST,n condition

Parameter:

condition See Tables 7-4 and 7-5.

Description:

LPTST is provided to allow conditional execution of statements in a loop. If the condition specified is true, the next n statements will be skipped. If it is not true, the next statement in sequence will be executed. If no n is specified, a default of one is assumed; that is, the next statement is skipped in true conditions. If n is 0, all the loops of a nest will be terminated not just the current one.

The FIND and NOFIND indicators are also set by the CSF command, depending on whether the request was successful or not (as indicated by the setting of bit 35), respectively. If no condition is specified on an LPTST command, the skip (or loop exit) will take place unconditionally.

Tests may be made for specified conditions. These conditions are described in Table 7-4.

Table 7-4. LPTST Conditions

Spec	Description
FIND	True if the last FIND, LOCATE, CHANGE, LC, or FC command matched a string, or if the last CSF command received a positive status.
NOFIND	True if FIND condition is false.
NEW	True if the current line is a newly inserted or altered image on this edit. Note that MOVEd and DITTOed lines are NEW, as are lines modified by INLINE and DOC, even if no changes were made to the line.
OLD	True if NEW is false.
ADD	True if the last image read by the ED processor came from an @ADD file.
NOTADD	True if ADD is false.

Tests may also be made for relational conditions on numeric or string values. The allowable values are described in Table 7-5.

Table 7-5. LPTST Values

Spec	Type	Format	Description
L	numeric	L	Loop counter for currently active innermost loop.
Ln	numeric	Ln	Loop counter for loop nested at depth n, where n is a single digit 0–9. 0 denotes the outermost loop, and a macro call counts as a loop level.
LD	numeric	LD	Loop nesting depth.
N	numeric	N	Current line number.
I	numeric	I	Current value of input line counter (as referenced by the GI command).
SP	numeric	SP	Line spacing for current line (value of 1 if the file being edited is not a print file).
LC	numeric	LC	Column number in which last string found by LOCATE or LC command starts.
CC	numeric	CC	Column number in which last string changed by CHANGE command starts.
LG	numeric	LG	Length of current text line (note that blank lines have a length of 1).
V	numeric	V	Value computed by most recent COMP command.
Xa	numeric	Xa	Value of variable set by COMP command, where a is void or is any alphabetic character.
QU	string	QU,x,y	Value is y characters starting at character x of the reply to the last "REM?" command. If y is omitted, 1 is assumed. If x and y are both omitted, the whole string is assumed. A value of zero for x will be treated as 1.
TX	string	TX,x,y	Value is a portion of the current line. The meaning of x and y is the same as for the QU specification.
MA	string	MA,n,x,y	Value is the parameter from the most recent call to the macro n. The meaning of x and y are as for QU. If x is explicitly entered as zero, the string will begin with the delimiter which followed the macro name on the macro call line. That is, the delimiter is treated as the the zeroth character of the parameter. An omitted value of x is treated as 1.
NN	string	NN	If the most recent uncompleted LOOP or macro call had a prefix (one of LN, IL, or NI), the value is a two-character string corresponding to that prefix (in upper case). Otherwise, the value is the null string.

The allowable numeric relationals are EQ, NEQ, LSS, LEQ, GEQ, and GTR. They represent equality, inequality, less than, less than or equal, greater than or equal, and greater than, respectively. If EQ or NEQ is specified, an additional clause of the form "MOD <m>" may be added following <n>, where <m> is a nonzero integer. This form tests the remainder on division of <counter>-<n> by <m> for zero or nonzero for EQ and NEQ respectively.

The only operators which are valid for string testing are EQ and NEQ. The value following the operator is a literal string. This literal string may begin with a quote (') character, in which case the string may include blanks and must be terminated with a quote (or by the end of the line). A quote may be included within a quoted string by writing two single quotes. If the string does not begin with a quote, it is terminated by the first blank encountered (or by the end of the command). The literal string may, of course, be computed with the LPSUB command.

LPTST may be used outside of a loop, with the result that the specified number of input lines will be skipped. This is most likely to be useful in batch mode editing or in an @ADD file. A message will be produced indicating the number of lines to be skipped, and, if in demand mode and not in an @ADD file, the skipped lines will be solicited with "SKP*" preceding the line number. The break keyin (@@X C) may be used to stop a skip; however, the next line will still be solicited with the 'SKP*' typeout, even though the line will not be skipped.

7.4.4. XTI Command

Purpose:

Allows a command to be typed in at a specific point in a loop.

Format:

```
XTI
XTII
XTI*
```

Description:

This command is intended for use in loops. It allows a command to be typed in at a specified point in a loop. When XTI is encountered while executing a loop, a single command will be solicited from the user and executed, following which the next command of the loop will be executed. This can be used if a number of similar but not identical changes are to be made to lines which are located by some complex sequence of commands. The locate operations may be done in the loop, with an XTI used at the point where the change is to be made, allowing the user to choose between the CHANGE, INLINE, or REPLACE commands.

The form " XTII " may be used instead. This form causes an unlimited number of commands to be read from the input stream. Reading of commands is terminated by a command of the form " XTI* ". If a loop contains a simple XTI command and it is desired to enter more than one command at this point, the XTII form may be used. Similarly, if it is desired to do nothing when an XTI or XTII is reached, XTI* may be entered.

7.4.5. LPEND Command

Purpose:

Indicates the end of the scope of a nested loop.

Format:

```
LPEND
```

Description:

The effect of LPEND when executed is to terminate the current loop iteration and return control to the statement following the corresponding LOOP command, or, if the iteration count is satisfied, give control to the statement following the LPEND. There must be one LPEND for each LOOP command with the exception of the outermost. It is not recommended that LPEND be executed conditionally or entered via the XTI sequence, as the results may not be what the user intended to achieve.

7.4.6. LPX Command

Purpose:

Sets the remaining iteration count for the current loop of a nest.

Format:

```
LPX n
```

Parameters:

n	Iteration count for the current loop.

Description:

The LPX command sets the remaining iteration count for the current loop of a nest to the value specified by "n". The default value assumed if n is omitted is 1. The count includes the current iteration. A value of 0 will have the same effect as " LPT,0 ". A value of 1 will cause the current loop to be terminated after the current iteration is completed. Therefore, a loop may be terminated with control going to the next outer loop by executing "LPX" and then a skip to the corresponding "LPEND".

Loops which do not stop on passing the end of file are identified by negative iteration counts. Therefore, if n has a negative value, the loop will be placed in non-stop mode. In particular, the command "LPX -1" may be used as the first command in a macro definition to indicate that the macro is not to be terminated if the end of file is passed; macros normally will stop if the end of file is passed.

7.5. MACRO Command

Purpose:

Defines a Macro.

Format:

```
MACRO n
MACRO* n
MACRO? n, n, n, ...,
MACRO?
MACRO??
```

Parameters:

n	Macro name.

Description:

For each of the first three forms, "n" denotes the name of a macro. Macro names are unique by the first three characters only. The first character must be alphabetic, but the other characters may be alphanumeric; the dollar sign may also be used. If a macro name duplicates the name of an existing ED command, the definition will be accepted, but the macro will never be callable. The first form specifies the entry of a macro definition. Macro definitions are entered exactly like LOOP definitions (including the use of @EOF and @EOF L), except that they are solicited by the typeout " MAC* ". A macro definition operation will destroy any stored loop, so that "LOOP!" will be invalid. The second form specifies the deletion of the definition of the specified macro. It is not necessary to delete a definition before redefining a macro; this is done automatically. This deletes only the in-core definition; a definition (as a n-ED-MACRO element) in a file is unaffected. The third form will list the text of one or more defined macros, the fourth form will list the names of all defined macros, and the fifth form will list the text of all defined macros.

Once a macro has been defined, it may be called by specifying its name as if it were an ED command. The remainder of the command line will be stored as the macro parameter, which may be retrieved by the LPSUB and LPTST commands. Macros may be thought of as loops which have an implied iteration count of 1. Macros may use loops and loops and macros may call macros, subject only to the limitation on loop nesting. Each macro call is an additional loop nesting level.

In addition to direct entry of a macro command followed by the text of the macro, a macro may be defined by implicit reference. If the ED processor encounters a command which it does not recognize, the input file, TPF$, and ED$PF (if assigned) will be searched for an element whose name is "n-ED-MACRO", where n denotes the command (up to three characters). If such an element is found, its text is loaded as the text of a macro with the name given as the command. The macro is then called. In this case, the @EOF L feature is not available. This mechanism allows the user to construct a library of ED macros to perform whatever functions are frequently required. This feature cannot be invoked while a loop or a macro is already active, just as it is impossible to define a macro or input a loop while executing a loop or macro. Macros defined in this manner which have not been called will not be listed by either the "MAC?" or "MAC??" command; these commands can only list macros which have been stored internally by the ED processor.

A useful example of a macro is the following "delete until locate" macro:

```
MACRO DUL
STK UP
BRIEF
LPT N NEQ 0
+
LOOP 99999
LPS MA,$,DUL,0,100*
L. $
LPT FIND
LPJ NOF
LPX
LPJ LPE
:NOF
D+
:LPE
LPE
STK DN
LPS NN,$
$(N)
@EOF
```

With this definition, the user may then enter "DUL ABC" and all lines will be deleted until a line containing ABC is reached.

Note that a macro with a void body may be thought of as a string variable. A new value is given to it by specifying the desired value on a call to the the macro, and the value may be retrieved with either the LPTST or the LPSUB command.

Because of the use of the prefixes LN, IL, and NI, a macro name may not begin with any of these pairs of characters. These prefixes may be used when invoking a macro; however, see the LPSUB substitution NN for application.

Saving MACROs in ED$MAC

The internal name ED$MAC is checked when the ED processor is called, when it terminates, and whenever a MACRO or LOOP is created or deleted. If this file is assigned when the ED processor is called, it will attempt to load a set of saved macro definitions in internal form from ED$MAC. At the other times mentioned, the ED processor will save all the known MACRO definitions, the current LOOP definition and the variables X, XA, XB,..., XZ in ED$MAC, if it is assigned to the run. Definitions saved in this format can be loaded much more rapidly than via the runstream or by the implicit definition method. By this means, it is possible to achieve continuity of MACRO definitions across several ED operations, irrespective of the nature of the intervening operations, so long as the assignment of ED$MAC is unchanged. If ED$MAC is not assigned to the run, even if it is an attached name, none of the above actions is taken. In other words, the ED processor will not create or assign this file; that is the user's responsibility.

7.6. USAGE CONSIDERATIONS

7.6.1. Searching Commands

The ED processor proceeds sequentially through the text. It is therefore more efficient to perform editing operations in a more or less sequential manner starting at the beginning of the text. Searching commands such as LOCATE and CHANGE require much computation and should be used sparingly; column limits may be used to speed the search.

7.6.2. Interrupts With the ED Processor

There are certain processes within the editor which if indiscriminately interrupted can cause the processor to fail. To protect against this, the processor is designed to stop only at specified points when it is safe to do so. If the user wishes to interrupt the processor, he may depress the the break key (or "MESSAGE WAITING" key) at any time. (This step is necessary only if the ED processor is printing at the time.) The system will respond with:

```
*OUTPUT INTERRUPT*
```

The user should answer with @@X C or @@X CO if he wishes to escape the current command. In the first case backed-up printout may follow before the interrupt takes place. If for some reason the editor's escape method is not satisfactory the user may enter @@X CIO twice. In this case the editor will return to edit mode, but integrity is not guaranteed.

The interrupt sequence will also have the following effects:

1. Backed-up commands on the same line as the one interrupted (when MCCHAR is in use) will be ignored.

2. If a LOOP or MACRO is in operation, it will be terminated.

3. If a LOOP is being entered, it will not be executed (but loop entry mode continues until an @EOF is encountered).

4. If an LPTST skip operation is occurring, it is terminated.

5. If an LPJUMP jump operation is occurring, it is terminated.

7.6.3. Filename Caution

Files with names of the form ED$xx (where x is any character) should be avoided since the ED processor uses such files internally.

7.6.4. Integer Expressions Instead of Integers

With the exception of the WAIT command and file/element cycle specifications, it is possible to use an integer expression anywhere an integer is permitted, such as for line numbers, column specifcations, et cetera. The expression must be one which would be acceptable to the COMP command and must in addition contain no embedded blanks. Note that this means that numbers entered with leading zeros are treated as octal.

As an example, the command

```
P N-3,N+3
```

would print the seven lines surrounding and including the current line.

If an integer expression is the first thing encountered on a line, it must begin with a sign, left parenthesis, or number in order to be recognized as an implied GO or NEXT command. (That is, L+3 is a call on the LOCATE command, not a GO to the third line after the line whose number is the current loop counter value.) If the expression begins with a sign, a NEXT is implied; if it begins with a digit or a parenthesis, a GO is implied.

7.6.5. Column Limits Immediate Specifications

In addition to the use of the LIMIT command, it is also possible to specify column limits for immediate use of a single command only, overriding the limits specified by default or by the LIMIT command. This applies only to the SEQ, LOCATE, CHANGE, and explicit printing commands, but not to commands which print a line implicitly because of alteration or transfer of file position. The syntax for specification of column limits may be any of those specified in Table 7-6.

Table 7-6. Immediate Column Limits Syntax

Format	Left Limit	Right Limit
[n,m]	n	m
[m]	1	m
[n,]	n	132
[,m]	1	m
[,]	1	132
[ ]	1	132

NOTE:

If the ED processor is locally reconfigured for 160 character lines, the values of 132 in the above table should be changed to 160.

Although the syntax of the immediate column limits specification is common to all commands which accept it, the position in which it must be specified differs. This will be described separately for each of the three classes of commands which allow immediate column limits.

7.6.5.1. LOCATE With Column Limits

For the LOCATE (or L) and LC commands, if immediate column limits are specified, they must immediately follow the command and any count specification, with no intervening blanks. At least one blank must separate the column limits from the LOCATE target string. For example, any of the following are acceptable:

```
L[3,5] ...
L.[3,5] ...
L,100[8,18] ...
LC*3[10,21] ...
```

Immediate column limits are not saved for later use on a LOCATE command with no target; they must be entered each time.

7.6.5.2. CHANGE With Column Limits

For the CHANGE (or C) command, immediate column limits must, if used, be the first specification after the string alteration pair, preceding the number of lines to be changed and any specifcation such as 'G', 'A', 'R', 'ALL', or 'REP'. For example, the following are acceptable:

```
C /.../.../ [5,10]
C /.../.../ [38] 5
C /.../.../ [39,] G
C /.../.../ [1,50] ALL
```

Immediate column limits, if given, are retained for later use with a CHANGE command with no specifications.

7.6.5.3. Printing Commands with Column Limits

Column limits may be used with the following printing commands: PRINT, P, PCC, QUICK, Q, OUTPUT, O, SITE, PUNCH, and CPUNCH. the immediate column limits, if specified, must be separated from the command (and its trailing delimiter, if any) by at least one space and must precede any line number specifications. For example, the following are all acceptable (the P command will be used for purposes of illustration, but any of the commands mentioned above may be used):

```
P [5,10]
P+ [11,21]
PI [38]
P [15,35] 10
P [39,] 11,20
```

Note that as for the LIMIT P case, column limits for printing will be adjusted to word boundaries. That is, a left limit is reduced to the next smaller number congruent to 1 modulo 4, and a right limit is increased to the next larger multiple of 4. For example, the limits pair [12,22] would have the same effect as the pair [9,24].

7.6.6. Default for F, FC, L, LC, and C Commands

For each of the F, FC, L, LC, and C commands, the previous target specification is saved and retrieved if the command is entered without any specification. F, FIND, and FC reference one saved string, L, LOCATE, and LC another, and C and CHANGE a third. For the C (and CHANGE) command, the saved string contains the number of lines, global, "REP", and "ALL" specifications (if any), as well as the old and new strings. For the F and L type commands, the saved string contains only the search argument, and different restrictions (such as number of lines scanned, column limits, or number of occurrences) may be used each time.

The commands may be entered in the form "F?", "L?", and "C?" . Each of these will print out the current default value to be used for the associated command. Other specifications on the command will be ignored. If no command of the associated type has been given, nothing will be printed.

7.6.7. LN, IL, and NI Feature

The characters "LN", "IL", or "NI" may be used to prefix all commands. If this is done, any lines of the file printed by the command will be prefixed with the associated line number, input cycle line number, or both, respectively. None of the three prefixes is considered to be part of the command for purposes of abbreviation (that is, "LNDITTO" may be abbreviated to "LNDIT" but not to "LND", since "LND" would be an abbreviation for "LNDELETE"). There are two special cases: "LNS" is an abbreviation for "LNSITE" (although "S" is not a valid command), and "LN" (rest of line blank) will print out the current and input line numbers and remain in edit mode. "LNn", "LN+n", "LN-n", "ILn", "IL+n", "IL-n", "NIn", "NI+n", and "NI-n", where n is an integer (expression) are all permitted. If "LN", "IL", or "NI" is used to precede the name of a macro, the macro will be called, and the LN, IL, or NI will be ignored. These pairs of letters cannot be used to begin the name of a macro.

7.6.8. Names for ASCII Control Characters

There are a number of commands which accept a single character as a parameter (TAB, CCHAR, LCHAR, MSCHAR, TCHAR, TCCHAR, and SHCHAR). For any of these, it is permissible to specify the name of an ASCII control character (such as ACK, BEL, DC3, DEL) instead of typing the character itself. Either BL or SP may be used to stand for the blank. The use of a name is also permissible instead of the octal code for the EXCH command. The STATUS command will type the name instead of the actual character if a control character is in use as a special character for the ED processor; this avoids adverse effects on the terminal in use, such as activating a paper tape reader.

7.6.9. Print File Operations

The ED processor performs certain special actions when editing a print file which do not occur when editing elements or general data files. These actions include:

- The new line created by the CHANGE, INLINE, or RETYPE commands will be given the spacing count of the replaced line, rather than a spacing count of 1.
- Lines transferred by the DITTO and MOVE commands will retain their previous spacing counts. Print control images included in the range to be transferred will be moved as well.
- If the ADD command is used to add a print file to a print file, the lines added will retain their spacing counts, and any print control images will also be added.
- If a new file (not element) is written by the SPLIT command, it will be created as a print file, with the appropriate label information, spacing counts will be preserved, and print control images will be written. The resulting file can be printed with @SYM just as if it had been created by the symbiont complex.
- Print files created by the ED processor (whether by processor call specification (as in @ED,U) or by the SPLIT command will have Fieldata/ASCII information in each image and each 224 word block will begin with an SDF image control word. These files will be compatible with all symbiont operations.

7.6.10. Edit Mode Commands in Input Mode

The ED processor allows any editing commands to be submitted while in input mode through the use of the CLIST$ mechanism. The format is as follows:

```
@EDIT command
```

where "command" denotes any command which is valid in edit mode. The letters "EDIT" must be in columns 2 through 5, and the command must begin in column 7. If the command is omitted, the ED processor will return to edit mode. This feature is convenient for setting up tab characters, AUTO counts, and so on when inserting a new element using the I option.

This format is acceptable in edit mode as well as in input mode, but it is superfluous. When the LOOP command is used and all or part of the loop is to be executed in input mode, @EDIT format may be used for any commands which are to be executed in input mode, such as LPSUB or LPTST.

This feature may be used when typing ahead (as in @@CQUE mode) to guarantee being in a particular mode. To force operation to edit mode, the user should enter "@EDIT"; to force operation to input mode, the user should enter "@EDIT INPUT". The desired mode will be entered regardless of which mode was previously active.

If the MCCHAR command is being used in input mode to indicate multiple lines on a single input line, each line which is to be interpreted as a command must have its own @EDIT following the MCCHAR character; without it, the command would be treated as input.

7.6.11. Character Command Processing

The commands CCHAR, MCCHAR, and SHCHAR define characters for which special action is to be taken on input. These characters are processed before the command on a line is analyzed, and then the character designated is deleted from the line. Thus, if the same character is specified twice in succession on any of these commands, the second time, the effect will not be what is intended. Since the character is deleted from the line, the command will appear to have no specification, which will deactivate the feature, and, if the MCCHAR character is involved, a blank line will appear to be present, switching the Editor's mode (unless in EOF mode) of the ED processor.

7.6.12. Reusability

The ED processor is reusable. Successive uses of the ED processor will not require an initial program load, which will conserve system resources. Note, however, that this means that the system log contains fewer entries for the ED processor than the number of actual distinct edits performed.

One consequence of the interaction between reusability and the command feature in input mode is that if a demand user attempts to terminate the ED processor with a transparent control statement (@MSG, @LOG, etc.), the editor sign-off message will be delayed until a nontransparent control statement (including @ED) is entered.

7.6.13. Restrictions and Limitations

The following restrictions and limitations apply:

1. If a FORTRAN data file is updated by the ED processor, the links used to hold backspacing information will be lost. Hence, the FORTRAN BACKSPACE statement should not be used on an updated FORTRAN file. Use of the R option with the ED processor will avoid accidental modification of the file.

2. Due to the internal structure of the ED processor, it is not possible to edit print files containing lines with spacing counts in excess of 63. Such lines will appear to be deleted.

3. The GI command may not be used when editing print files, since the necessary information is not available, due to the presence of print spacing information.

7.6.14. The ED$TC File

The ED processor attaches the internal name ED$TC to a file whose name is project-id*ED$TCrun-id. For demand users, this file will normally be a catalogued file; if another run with the same original run-id is active and using the ED processor or for a batch run, this file will be a temporary file. ED$TC (if catalogued and not temporary) is assigned with the D option. This allows the implementation of the AUTO RECOVERY feature. If a run terminates normally, ED$TC will be deleted, and there will be no auto recovery. If a (demand) run terminates abnormally, such as by system crash, line drop, terminal timeout, @@TERM statement, or operator keyin (SM site-id T), the ED$TC file will be retained. This permits auto recovery when a new run is initiated and the ED processor is executed.

NOTE:

The new run must have the same project-id and run-id as the run which terminated abnormally.

If a user fails to start such a run and use the ED processor in it, the ED$TC file will remain catalogued indefinitely. Therefore, a user should be aware of this situation and delete a file which is no longer required.

The internal structure of the ED$TC file is not of importance to most users; however, since it contains certain link addresses which depend on the internal structure of the ED processor, the data in ED$TC must be verified to be correct. This is done through the use of a validity constant (VALCON). The value of VALCON will be different for different levels of the ED processor, so ED$TC cannot be used between levels of the ED processor. VALCON may also be used to detect possible corruption of ED$TC due to hardware errors, and a change in this value is also possible if the ED processor itself has a software failure. If a VALCON error occurs, it is necessary to erase ED$TC before calling the ED processor again.

7.6.15. Obsolete Commands

A number of commands are still defined in the ED processor which are now considered obsolete and are thus not documented. These are listed in Table 7-7.

Table 7-7. Obsolete ED Processor Commands

Command	Description
BOT B	Same as APPEND.
BRIEF BR	Same as "ON B".
CLIMIT	Same as "LIMIT C".
END	Same as OMIT.
GO n	Goes to line n of file, where n is an expression.
HEAD H	Same as "1" (goes to first line of file).
NEXT n N n	Moves forward n lines in the file (backward if n is negative).
ON n	Same as "P+ n".

Table 7-7. Obsolete ED Processor Commands (continued)

Command	Description
PLIMIT PL-	Same as "LIMIT P".
SAVE	Same as "LIMIT C".
TOP T	Same as "0" (goes to top of file).
VERIFY V	Same as "OFF B".

8. Procedure Definition Processor (PDP)

8.1. INTRODUCTION

The Procedure Definition Processor (PDP) accepts symbolic input defining Assembler, MASM, FORTRAN, or COBOL procedures and builds an element in the user-defined program file. These procedures may subsequently be referenced in an assembly or compilation without definition.

8.2. @PDP FORMAT

Purpose:

Places entries in the Assembler, FORTRAN or COBOL procedure table in a program file table of contents (see 11.2.1.2 for a description of these tables). The entries are put into the table of contents of the symbolic output file specified by the user. If no symbolic output is produced during the execution of PDP, no procedure table items are generated. These entries contain labels in Assembler or MASM procedures defined as PROC entry points and names which are used to call FORTRAN or COBOL PROCs. When a call is made for a PROC in a source program, the language processor automatically retrieves the PROC using the System Relocatable Library routine (BSP$) to search the table of contents for the PROC name. If more than one PROC of the same type and the same name are contained in a program file, the search by BSP$ will point to the last PROC entered with that name.

PDP is called by the @PDP control statement.

All parameters in the @PDP control statement are optional except eltname-1.

Format:

```
@label:PDP,options    eltname-1,eltname-2
```

Parameters:

options	See Table 8-1.
eltname-1	Normally specifies the input element. However, when the I option is specified, eltname-1 specifies the new program file element.
eltname-2	Specifies the output element.

Table 8-1. @PDP Control Statement, Options

Option Character	Description
A	Accept the results as correct even if errors are detected.
C	Indicates a COBOL procedure element.
F	Indicates a FORTRAN procedure element.
I	Insert a symbolic element into program file from the control stream.
L	Produce a complete listing of the output element with line numbers.
M	Indicates an 1100 Series Meta-Assembler (MASM) Procedure element.
N	Suppress all listings.
S	Generate a single-spaced listing of the output element.
U	Generate a new cycle of the symbolic element.
W	List correction lines if corrections are provided.
X	Take ERR$ exit from PDP if errors are detected.

NOTE:

The source input routine options (see Table 1-2) also apply.

In the absence of the F, C or M options, PDP assumes that it is inserting or updating an Assembler procedure element.

Cycling of procedure elements is permitted. The cycle number may be increased if the U option is specified. When a procedure is included in an assembly or compliation, the procedure from the latest cycle of the procedure element is supplied.

Examples:

```
1.  @PDP,L          A.B,C
2.  @PDP,L          A.B,,C
3.  @PDP,L          A.B
4.  @PDP,I          AFILE.PROS/AB
5.  @PDP,U          BFILE.PAT/DE
6.  @PDP            AF.PR1,BF.PR2
7.  @PDP,ULF        D.FORPROC,E.FORPROC
```

1. Generates a procedure element from file A, element B and places the new element C in TPF$. Generates a complete listing.

2. Generates a procedure element from program file A, element B, calls it element C, and places it in program file A. Generates a complete listing. Eltname-2 must not name a tape file.

3. Generates a complete listing of element B from file A. No procedure entries are made.

4. Procedure definitions following this @PDP control statement are placed in file AFILE as element PROS, version AB, cycle 0.

5. Corrections are made to element PAT, version DE, latest cycle of file BFILE to generate an updated cycle of the same element in the same file.

6. Corrections following the @PDP control statement are merged with the most recent cycle of element PR1 in file AF to generate cycle 0 of element PR2 in file BF.

7. Produces an updated cycle of the FORTRAN PROC element FORPROC in the file E using as input the element FORPROC in file D. All element cycles are retained in element FORPROC up to the cycle maximum.

PDP generates certain flags on the output listing when it detects error or other conditions. These flags appear on the output listing between the line number and the text of a line. Table 8–2 describes the flags.

If PDP detects any errors, entries are not made in the assembler, FORTRAN or COBOL procedure tables unless the A option is specified. If errors are detected, and neither the W nor A option was specified, no symbolic output is created. Except for the errors shown in Table 8–2, PDP does not detect language processor syntax errors within PROCs.

When a PDP symbolic is transferred to an element file (on tape) using the FURPUR statement @COPOUT, the procedure table entries are carried along with the element. On the @COPIN of a PDP symbolic, FURPUR puts the procedure table items of that element into the table of contents of the program file.

When a PDP symbolic is transferred from one program file to another program file using the FURPUR statement @COPY,P, the procedure table entries are carried along with the element. Therefore, it is not necessary to reprocess a PDP symbolic element using PDP if the element has been brought into a program file using the FURPUR statements @COPIN or @COPY,P.

When a single procedure element is transferred from one program file to another program file using the FURPUR statement @COPY,S, FURPUR transfers only the latest cycle of an element having more than one element cycle. The procedure table entries belonging to this element are put into the destination file's table of contents. However, these entries will be incorrect if the element in the file of origin contained deleted images. Therefore, it is preferable to use PDP to transfer a single procedure element from one program file to another program file. See 11.2.3.3 for more information on the structure of a symbolic element.

If a symbolic PROC element is updated with the symbolic output in the same file as the symbolic input, PROC names which appear in the symbolic input but not in the symbolic output will not be marked as deleted in the procedure table in the program file table of contents, even though the element in which they appear is marked as deleted in the element table. A @PACK of the file following the PDP processing will remove the deleted element along with the corresponding PROCs.

Table 8-2. PDP Flags

Flag	Meaning
-	This symbolic is not an error flag. It indicates that the line on which this flag appears is a continuation of the previous line by virtue of a semicolon appearing on the previous line. Assembler and MASM procedures only.
E	This flag implies the existence of an expression error within the label field of the current line. This is set as a result of an illegal label symbol or sequence of symbols. Due to the complexity of the form which the operation field may assume, no attempt is made to detect expression errors within this field. No entry point will be established for a line whose label field contains an expression error.
I	An illegal operator in an assembler procedure is indicated by this flag. This can only occur when a DEF directive has been encountered and anything other than EQU, EQUF or FORM directives or FUNC definitions occur between the DEF line and a PROC directive. In addition, a message is printed indicating that the previous DEF directive was ignored. Assembler procedures only.
L	This flag indicates that an error has occurred in the procedure level sequence. This is caused by an excessive number of END cards such that one is encountered when the procedure level is zero. The level is left at zero, but the L flag is set.
T	This flag is generated when a label or operation field exceeds 12 characters in assembler or MASM procedures, or the label on a PROC statement in FORTRAN or COBOL PROCs exceeds 12 or 30 characters respectively.

9. File Administration Processor (SECURE)

9.1. INTRODUCTION

The SECURE processor protects the physical security of catalogued files, which reside on mass storage, by producing tape backups.

The text of files on mass storage may be destroyed either inadvertently through system failure or user error, or purposely to reduce overcrowding of facilities or to remove certain mass storage units from the available facilities pool. In any case of purposeful destruction, the presence of a current backup must first have been assured. Because a file may be inadvertently destroyed and the latest backup may not be current, a record must be kept of the file's memory lapse. Memory lapse is defined as the time period that starts at the first updating after the latest backup was created, and ends with the recovery of the file from the backup copy. This is the period of time during which any additions or deletions were not retained.

When a file's text on mass storage has been destroyed and the backup is the only available copy, the file must be marked unloaded, so that an automatically initiated load of text back to mass storage occurs when the next attempted assignment of the file is made. The run which makes the @ASG request that forces this load may be held in a wait status until the load is completed.

The process of selecting files as potential candidates for unload when some number of currently used tracks must be vacated and made available for new allocation, selects those files which will probably be the last to be reassigned, in order that the files actually unloaded can be left dormant for as long as possible before they have to be reloaded. There is sufficient flexibility in the formulation of the unload mechanism to permit qualified onsite personnel to make dynamic adjustments to the individual weight attached to each of the variables which go into this unload eligibility factor determination.

An unload inhibit option is defined for use by certain files which cannot be removed from mass storage due to real-time or other needs. There is also an even more restrictive guard option which inhibits even the privileged read necessary to make backup copies. The guard option is required for certain special files which are internal to the system, highly transient, or highly classified.

In addition to the basic SAVE, UNLOAD, and LOAD commands, there are supporting commands to register unknown files recorded on backup tapes made at another SPERRY UNIVAC 1100 Series Systems site, and to list the memory lapses that have occurred for a file. It is also possible to allow all commands to be selectively directed, when desired, toward only certain named files, projects, accounts, qualifiers, tapes, or mass storage units.

Finally, it is possible for the SECURE processor to assist in a catalogued file recovery process. This depends on a tape copy checkpoint of the master file directory (MFD) and the entire set of file backup

tapes to restore the set of catalogued files to a state at least as current as the time the MFD checkpoint was made. A file is considered restored when MFD items can be retrieved from mass storage and its text can be retrieved from mass storage or tape.

9.2. MAJOR FUNCTION DEFINITIONS

The SECURE language contains six major commands:

1. SAVE creates a duplicate copy on tape of both the MFD information and the text of specified catalogued mass storage files.

   SAVE, and all other actions of the SECURE processor are not performed on files which were catalogued with a G (guard) option on the @ASG or @CAT control (see Volume 2-3.7.1 and 2-3.7.3) statement. The purpose of the G option is to override the privileged mode capabilities of the SECURE processor for particular files. Several of the Executive's internal files, including the scheduling file, the accounting file, and the symbiont files use the G option.

2. UNLOAD implies SAVE unless there already exists a tape backup copy of the file that was made by SAVE after the last write was done on the file. UNLOAD then releases the space occupied by the text of the file on mass storage and updates the MFD to mark the file unloaded. UNLOAD, or any other action dependent upon unloading the file, is not performed if the file was catalogued with a V (unload inhibit) option. This command is not meaningful for removable disk files as their text is never marked as unloaded.

3. REMOVE implies UNLOAD, unless the file is already unloaded. REMOVE then causes the file to be decatalogued from the MFD.

4. REGISTER scans the tape reels named in source language which contain SECURE-produced backup copies of mass storage files, and catalogues as unloaded those files that are not currently catalogued. This command can also be used to restore the MFD items for an entire set of catalogued files by using the 'MFD snapshot' tape.

5. LOAD operates on currently catalogued files that are marked unloaded or on currently non-catalogued files if a tape is designated as the source of input. LOAD copies the text of the file back to mass storage and turns off the unloaded indicator in the file's MFD entry.

6. LIST produces information based on the contents of the MFD.

9.3. @SECURE CONTROL STATEMENT

All parameters on the @SECURE control statement are optional.

The format of the @SECURE control statement is:

```
@label:SECURE,options  eltname-1,eltname-2
```

options — See Table 9-1.

The eltname-1 and eltname-2 parameters name the symbolic input and output elements, respectively; see Volume 2-3.9 for rules governing their use.

Table 9-1 describes the options that can be specified on a @SECURE control statement. In addition to these options, the source input routine (SIR$) options (see Table 1-2) may also be used.

Table 9-1. @SECURE Control Statement Options

Option Character	Description
A	Do not take error exit even if errors are detected.
B	Do not do exclusive assignments of files to be acted upon. This option should be used with care.
C	Enable the checksum feature in the SECURE processor to compute and write to tape a checksum total for each text block written by a SAVE command or use this value as a check when data is transferred from tape during a LOAD command or from tape to tape during a SAVE ALL operation (see 9.14.1).
D	Include the text and directory of removable disk files on SAVE operations whether system-wide or by project, account, or qualifier. Removable disk files may be copied to a tape set unique from that of mass storage files. See SPERRY UNIVAC 1100 Series Executive System Operators Reference, UP-7928 (current version).
E	Display directory error diagnostics for all files that cannot be processed. Error diagnostics will be displayed for all files encountered by SECURE that have directory errors or inconsistencies that would prevent the files from being processed. The format of these diagnostics is: MFD-ERROR-*nn* ON FILE *qualifier*filename(fcyc)* where *nn* refers to the following error types: 0 — Main item extension chain is incomplete. 1 — Main item identifier bits incorrect. 2 — Invalid character as Qualifier, File name, Project or Account. 3 — Main item extension identifier bits incorrect. 4 — Main item extension Qualifier*Filename does not match that in the Main item. 5 — Sequence identifier in Main item extension is invalid. 6 — Lead item identifier bits incorrect. 7 — Lead item Qualifier*Filename does not match that in the Main item. 8 — Lead item sector 1 identifier bits incorrect. 9 — Excessive DAD items. 10 — No valid look-up entry for this Main Item. 11 — Missing Search Item for this Main Item. 12 — Main item extension linkage corruption. 13 — Lead item linkage corruption. 14 — Lead item extension linkage corruption. 15 — DAD item linkage corruption. 16 — Qualifier*Filename does not match that returned by DREAD. 17 — Incomplete pack information on a Register of a removable disk. 18 — Incomplete reel information on a Register of a cataloged tape. 19 — Incomplete lapse information. 20 — Incomplete Backup Reel information.
F	Signifies user is operating on 'filename only' level. Allows SECURE to avoid doing a full-system directory snapshot when processing only files specified by name. This option may not be used when referring to catalogued tape files or removable disk files in source language, nor may it be used when limiters other than FILE(S) are used in source language. This option should not be used with large numbers of named files.

Table 9-1. @SECURE Control Statement Options (continued)

Option Character	Description
L	Produce the most comprehensive printed listing.
N	Suppress all listing except error diagnostics.
O	Scan all files, and print a summary listing including only files marked 'SECURE' or 'hardware' disabled.
R	Scan all files marked disabled and print summarized results. Do not process any source language. @SECURE,R is called from within the Executive following a recovery bootstrap, or may be called by the normal user from within a non-privileged run. In non-privileged mode, no projects or accounts are printed in the summary listing.
S	Produce a summary printed listing.
T	Specifies that directory items for catalogued tape files are to be copied to both the SAVE created backup tape and the 'MFD snapshot' tape. Without this option, a REGISTER of the directory tape is always necessary when performing a REGISTER of catalogued tape files. It should be noted that the text of a catalogued tape file is never saved by SECURE.
X	Take error exit if all specified tasks cannot be processed.
Z	Designates that directory information for files being decatalogued by REMOVE operations will be retained in a catalogued file, ARCHIVE$, to be used in the future by SECURE to recreate the REMOVEd files (see 9.14.5).

9.4. INPUT AND OUTPUT BACKUP TAPE ASSIGNMENTS

Users calling the SECURE processor for tape operations should first assign tape units for input and output of backup tapes by means of control statements with the following format:

```
@ASG,NTF      OBACKUPnn,T
@ASG,NT       IBACKUPnn,T
```

The @ASG,NT control statement causes the tape unit to be assigned temporarily and with the initial tape load message suppressed. nn is an optional one- or two-digit number from 1 to 63 used to distinguish between multiple OBACKUP or IBACKUP names. If nn is omitted, 1 is assumed. The F option on the OBACKUP assignment is necessary to avoid a filename check when a read is attempted on the tape if labeled tapes are in use.

All SECURE processor operations involving either tape reading or writing can be performed on tape units assigned as output backup (OBACKUP[nn]). As a safety feature to help protect the contents of existing backup tape reels, only reads are performed by the SECURE processor on tape units that are assigned as input backup (IBACKUP[nn]).

Some examples of these assignments as they might appear in a single run are as follows:

```
@RUN
@ASG,NTF      OBACKUP24,T
@ASG,NT       IBACKUP7,T
@ASG,NT       IBACKUP8,T,4832A/4432C/4162A
```

If the operations to be processed do not require the predefinition of specific tape reel numbers that would otherwise be unknown to the SECURE processor, the user need not be concerned with identifying them. For example, the SECURE processor dynamically requests as many new blank tape reels as necessary when creating new backups and records the reel numbers used in both the MFD item of the backed-up file and in the user's printed output listing. As another example, the SECURE processor automatically consults the MFD item to get the correct reels necessary to load the text of an unloaded file back to mass storage.

There are instances, however, when a specific sequence of numbered reels should be associated with a particular OBACKUP or IBACKUP tape unit. These associations are specified by giving a reel list on the @ASG card or by source language statements of the format IBACKUPnn = reel list or OBACKUPnn = reel list, as follows:

```
@SECURE,IS
 IBACKUP7 = 95
 IBACKUP8 = 4458, 4461, 4462
 OBACKUP24 = 701, 702, 703
```

As an example, the SECURE processor allows a set of reels in backup format to be registered with the Executive, so that any files previously saved to these reels, but which are not currently catalogued, are recatalogued as unloaded mass storage files. Following a reel number association like that done above for IBACKUP8, the REGISTER command could be invoked by the single control statement: REGISTER FROM IBACKUP8.

When a particular SECURE operation involves the transfer of many text blocks to and from tape, it is possible for SECURE processor to initiate multiple, concurrent I/O operations to optimize efficiency by assigning several output tape units. The multi-activity operational procedures are described in detail in 9.13.

If no usable backup tape units are assigned at the time that the SECURE processor determines that tape I/O must be done, the SECURE processor will assign a single unit using the system standard type and density.

If an IBACKUP tape unit is not assigned prior to a REGISTER DIRECTORY TAPE FROM IBACKUP operation, SECURE will dynamically assign it. Then, immediately after the tape has been read, IBACKUP will be dynamically freed (@FREE).

The most common cases where the SECURE processor will need tapes are the periodic SAVE commands needed to generate a set of new backups and the LOAD of an unloaded file which some run is attempting to assign. It is not necessary for a site to keep a tape unit available at all times just to enable SECURE to handle these two common cases. The SECURE run is placed in a facility wait state until the tape unit becomes available.

System-initiated UNLOADs to relieve overcrowding of mass storage can normally be accomplished without creating new backups (or requiring tape unit assigns). However, if tape assigns are necessary and no tape units are available, the SECURE processor initiates a console message informing the operator that a tape unit must be made available. If system conditions warrant the operator may make a tape unit available by either restoring a tape unit that is in a reserved or downed state or by terminating another run (by using a checkpoint, E, or X keyin) that has a tape unit assigned.

9.5. CATALOGUED FILE ASSIGNMENTS

The SECURE processor performs no actions on files which are actively assigned to another run at the time that SECURE is executing, unless those files were catalogued as read only. Files to be operated on by the SECURE processor are dynamically assigned with an exclusive-use option to prevent other runs from writing on current write-enabled files.

If the number of runs in the system is temporarily reduced at the time a SECURE processor SAVE run is called, the problem of files not getting backed up, due to use by other runs, is minimized. To further reduce this problem, SECURE attempts a second or third dynamic @ASG,AX of those files that were busy during the first pass. The SECURE listing reports any files which could not be saved.

9.6. PRIVILEGED MODE OPERATION

A run in privileged mode indicates that a program within the run (such as the SECURE processor) can access the text and full MFD information for all files catalogued in the system that do not have a G (guard) option inhibit on them, without supplying any of the keys and without regard to whether the file might be catalogued as private or read only or write only. This permits the SECURE processor to initiate I/O and other operations on a file that are necessary to create backups, or to delete or restore text.

One of the Executive interfaces which checks to see if a run is privileged is the MSCON$ request. This request allows direct reading and altering of MFD items. The MSCON$ request which gets a copy of the entire MFD and writes it into a user-specified file, provides an example of the distinction between privileged and nonprivileged runs; if the run is privileged, the MFD is copied unchanged; if the run is not privileged, MSCON$ obscures all project-ids, account numbers, (other than those of the calling run), and all keys.

The operations which a user can direct SECURE to perform when the run is not privileged are summarized in 9.9.

9.7. SECURE SOURCE LANGUAGE

SECURE employs a source language structure for input which gives the user a simple, but flexible, format for calling on the processor to perform any or all of the allowed functions. Basic source language components are ordered as follows:

```
command ALL limiters name-list EXCEPT name-list;
FROM equipment-name TO equipment-name
```

All parameters are optional except command. Spaces are required between fields on the source line, Δ . Δ allows comments and a ; specifies continuation.

9.7.1. Standard Commands

The commands recognized for the SECURE processor are:

SAVE
UNLOAD
REMOVE
REGISTER
LOAD

LIST REELS namelist
LIST FILES namelist
LIST UNLOADED FILES
UEF +nn namelist
UEF -nn namelist
REVERT filename
LIST LAPSES namelist
CLEAR LAPSES namelist
CONSOLE
NO-MFD
DELETE
ARCHIVE
MERGE ARCHIVE
END

Unless the word ALL is used, the SAVE command causes saves to be done only on those files which do not have a current backup. A backup is current depending on whether any write operations have been performed on the file since the time the last backup was made. However, when the word ALL is used, the SAVE command causes all files in the namelist to be saved regardless of whether a current backup exists. Because the SAVE ALL command causes a tape to tape copy if a file is unloaded, an @ASG,TN IBACKUP,T is necessary in addition to the normal OBACKUP assignments for complete execution. However, the IBACKUP assignment should not be included if it is intended only to save the currently loaded files. When the word DIRECTORY is stated after the SAVE, only a copy of the current 'MFD snapshot' tape will be produced.

If PACK or PACKS is specified after SAVE, text as well as directory information of the files residing on the removable disk pack(s) specified in the following namelist will be copied to tape. If the word ALL is used instead of a pack-id namelist, all removable disk files currently catalogued in the MFD will be included. A backup of the text of removable disk files is created when its filename or the pack-id on which it resides is specified in source language.

The SAVE ALL WITH RECOVERY command is used to finish an incomplete SAVE ALL operation, starting at its point of abnormal termination. Any file subset designators on the initial SAVE ALL statement must be duplicated on this statement also. On any output operation requiring a SAVE ALL (i.e., SAVE ALL, UNLOAD ALL, REMOVE ALL), SECURE will catalogue a file, SYS$*RECOVERY, at the outset. If the operation terminates normally, this file will be decatalogued. The file remains catalogued if the operation terminates abnormally. If a subsequent SAVE ALL WITH RECOVERY operation is done, only files from the subsets specified that have not had backups created since the catalogue time of SYS$*RECOVERY will be saved (SAVE).

A file's disabled status does not prevent its being maintained by SECURE. Any 'disabled' bits set in a file's main directory item will be preserved by REGISTER. The LOAD action will be attempted on any unloaded file, regardless of disabled status. Failure to complete the loading of text normally will result in the file's being marked disabled (FAC reject status bit 6). SAVE will copy the text of a hardware disabled file only if no backup exists. SAVE ALL, operating on a hardware disabled file, will copy the text from the backup tape, if one exists. All other SECURE actions will ignore a file's disabled status.

Unless a namelist is given or the word ALL is used, the UNLOAD command does not cause more files to be unloaded than is necessary to free 3000 tracks on mass storage. The particular files chosen for unload, in this case, are selected using procedures explained in 9.8. To change the preset limit of 3000 tracks to some other value, the UNLOAD specification may be stated as UNLOAD TRACKS = nnnnnn. If a namelist is given all files so specified are unloaded, regardless of how much or how little space they occupy on mass storage. If the word ALL is used, all catalogued files are unloaded. If unloading only position granular files is desired, the command UNLOAD POS = nnnnnn is used, where nnnnnn is the number of positions.

The REMOVE command deletes from the MFD the files indicated by selection specification (or all catalogued files if no specification is given) after the presence of a current backup copy for each has been assumed. The keyword ALL causes SECURE to create a new backup copy even if a current backup exsits.

The REGISTER command requires the addition of a FROM IBACKUP to which reel numbers have been associated. Unless a namelist is given, the REGISTER command operates on all files found on the equipment associated with IBACKUPnn. REGISTER causes cataloguing of the found files using the attributes of the file as found on the backup tapes. Such files are marked UNLOADED. No SECURE operations except another REGISTER FROM IBACKUP may be used if this command is stated in source language.

The LOAD command causes the text of unloaded files given in a namelist (or if ALL is specified, every catalogued file) to be copied from backup tape to mass storage. Use of UNTIL UEF = nn or UNTIL PERCENT = nn with this command enables the loading of some subset of all the catalogued files. Files are sorted in ascending order by unload eligibility factors (UEF's) and are loaded until either the desired UEF cutoff value is reached or until the desired percentage of available mass storage has been filled. The command LOAD...FROM IBACKUPnn combines the REGISTER and LOAD actions in one operation. If the file is found to be already catalogued, but marked as unloaded, the file's text is loaded if its MFD designates this tape as its backup copy.

The LIST REELS command produces a summary listing of the current set of backup tapes sorted in ascending order by reel number. This command must be in a separate SECURE execution with no other source language present.

The LIST FILES command displays the same information as the LIST REELS, but entries are sorted alphabetically by qualifier and filename. This command must also be specified individually in a SECURE execution.

LIST UNLOADED FILES creates the same-formatted listing as LIST FILES, but includes in its summary only those files whose text is currently marked as unloaded. This command must be in a separate SECURE execution as well.

The UEF +nn or UEF -nn commands may be used to selectively add or subtract a number nn from the computed unload eligibility factor (UEF) for the named files, projects, qualifiers or accounts. This factor determines the order in which files are unloaded when mass storage space becomes crowded. The UEF bias remains in effect only for the current execution of SECURE. The larger the UEF is for a given file, the more eligible the file is for unloading. See 9.8 for selection of files for UNLOAD.

REVERT filename means revert to a previous backup copy of the named file. This can be used when a user accidentally overwrites the latest copy of the file on mass storage and wants the text in the existing backup to be retained instead of the more recent text now residing on mass storage. When the text of files on mass storage is inadvertently destroyed, the backup tape becomes the primary copy. If, however, the backup tape is not current, a record is kept of the time period during which any addition or deletions to the mass storage file were not retained. This record represents a file's memory lapse. Any number of these may possibly occur over the life of the file.

The LIST LAPSES command provides a printed listing of the memory lapse entries for all files or for the set of files specified by the namelist. This command must be in a separate SECURE execution.

The CLEAR LAPSES command requires a namelist and erases the record of any existing lapse entries in the set of files specified by the namelist.

The CONSOLE command allows the user to change source language input modes from card images to console keyins when desired. When this command is encountered in source language, 'ENTER SECURE COMMAND' appears on the onsite console, and the user may key in the action desired. If

an 'END' directive is received while in this mode, all remaining SECURE source language from the runstream is processed.

The NO-MFD command suppresses the creation of a directory tape by a privileged run.

The DELETE command removes files without ensuring a current backup exists.

The ARCHIVE command is an extension of the REMOVE command. In addition to deleting the file, selected information is retained in SYS$*ARCHIVE$ so that the the file may be subsequently restored.

The MERGE ARCHIVE command consolidates a set of backup tapes holding Archived files. If a tape list is given files which are wholly contained on the specified tapes will be copied, without a list all Archived files will be merged to a single tape set.

END is an optional terminator to mark the end of source language statements.

9.7.2. Namelist and Limiters

Namelists are strings of file, project, qualifier or account names which designate particular file sets. Unless otherwise specified, all commands allow any of the namelists below. They are preceded by an appropriate identifier as follows:

FILE(S) filename-namelist

PROJECT(S) project-namelist

ACCOUNT(S) account-namelist

QUALIFIER(S) qualifier-namelist

Names in the list are separated by commas. Actions specified in source language are limited to the set of files specified in a namelist. If no namelist is used with a particular command, a distinction must be made. If the run is not privileged, it is assumed that the action is to be applied only to those files under the user's project-id. Exceptions to this general rule are given in the description of each command (see 9.7.1). A limiter may be specified to restrict the file set described in the specified or implied namelist to files within the named category.

A limiter may be used to restrict the file set described in the specified or implied namelist to files within the named category. The self-explanatory limiters are: PUBLIC, PRIVATE, READ-ONLY, WRITE-ONLY. Several other limiters are only meaningful when used in conjunction with specific commands:

- BEFORE nn DAY(S) or MONTH(S) and AFTER nn DAY(S) or MONTH(S) refer to the time since a file has been referenced, and should only be used with the SAVE and REMOVE commands. These allow separation of a file set into current and dormant categories based on the time spanned since the last reference to the file. The statement, SAVE ALL AFTER 30 DAYS, would only include files in the SAVE ALL which were referenced within the last 30 days. By substituting BEFORE for the AFTER, a SAVE ALL would be only done on files with reference times 31 days or older.

- TAPES is a limiter to be used only with the SAVE and REMOVE commands. When this is used, only files marked as unloaded and backed up on the tape(s) specified will be included in the SAVE or REMOVE. An example of its usage is: SAVE (or REMOVE) TAPES 1, 2, 3.

- The PACKS limiter denotes that only files catalogued on the removable disk pack-id(s) specified are to be included. This limiter may be used in conjunction with all commands except the LOAD, UNLOAD and REGISTER.

- ARCHIVE is a limiter to be used only on the LIST FILES (or REELS), REMOVE and LOAD FROM IBACKUP commands, and designates that the contents of the file SYS$*ARCHIVE$ are to be processed.

9.7.3. Exclusions

EXCEPT preceding a namelist, causes those files, projects, qualifiers or accounts in the list to be excluded from the particular action.

9.7.4. Direction

When an action is to be directed to or from particular tape files or mass storage devices, a FROM or TO designator is used from the following set, where sss/uu are mass storage subsystem/unit numbers:

FROM IBACKUPnn

FROM OBACKUPnn

FROM UNIT(S) ss/uu-list 1, ss/uu-list-2,..., ss/uu-list n

TO UNITS(S) ss/uu-list 1, ss/uu-list 2,..., ss/uu list n

TO PACK *pack-id-1, pack-id-2..., pack-id-n*

TO FIXED

Note that it is not possible to move files from one mass storage unit directly to another using FROM and TO.

For files stored on dual subsystem devices, only the primary subsystem number is recorded in the master file directory for such files. Therefore, any subsystem references made in SECURE source language must refer to the primary subsystem number of a dual subsystem. References to the alternate subsystem of a dual subsystem will not be successful.

With the introduction of EXEC level 35 I/O configurations, subsystem/unit specifications will be replaced by a list of device mnemonic names.

Example:

FROM UNIT D30TU4,D30TU5

The two directives TO PACK and TO FIXED, can only be associated with the REGISTER (or LOAD) FROM IBACKUP commands when they are used to catalogue removable disk files from a backup tape. If TO PACK is specified, any removable disk files found on the backup tape will be catalogued with the specified pack-id. If TO FIXED is used, the files will become fixed disk files. In either case, the equipment type of the files will not change.

9.7.5. Examples of Source Language

The following are examples of source language specifications:

```
IBACKUP = 2350, 2351, 2352
OBACKUP = 4451, 4452, 4453
SAVE ACCOUNT 399125 EXCEPT FILE MY*FILE
SAVE ALL PROJECT MERCURY TO OBACKUP
UNLOAD TRACKS = 1500 FROM UNIT 12/3
REMOVE PROJECT SATURN TO OBACKUP
LOAD FILES FILE1, FILE2, FILE3
REGISTER EXCEPT ACCOUNT 399126 FROM IBACKUP
LOAD PROJECT VENUS FROM IBACKUP
UEF - 10 FILE FREQUENTLY*USED
REVERT FILE MY*ERROR
LIST LAPSES
CLEAR LAPSES PROJECT MERCURY, SATURN
LIST REELS
REVERT FILE MYFILE
SAVE ALL AFTER 20 DAYS
SAVE ALL TAPES 1, 2, 3
SAVE ALL PROJECT MARS WITH RECOVERY
LOAD PACK REM001 FROM IBACKUP TO PACK REM002
REGISTER FROM IBACKUP TO FIXED
LIST FILES ARCHIVE
END
```

9.8. SELECTION OF FILES FOR UNLOAD

If an UNLOAD command is given without naming any particular files, projects, accounts, qualifiers or mass storage units, it is assumed that the SECURE processor has the responsibility to scan the entire set of catalogued files and to select the subset that must be unloaded to acquire the additional assignable mass storage space that is needed.

This differs from the action of the SECURE processor when particular files, projects, accounts or qualifiers are named for unload, in which case all eligible candidates in the named set are unloaded. This is also in contrast to the action of UNLOAD ALL, where all eligible files in the system are loaded.

When the SECURE processor is called to do saves to backup tape, drum-to-tape I/O is unavoidable. When the SECURE processor is called to do only an unloading operation, however, it is sometimes possible to avoid tape I/O. This situation results when there exists an adequate reservoir of current backups of unload-eligible files already out on tape to meet the requirements of a general UNLOAD command.

Often, at the same time that mass storage facilities become overcrowded, other system resources become inadequate, making it necessary to temporarily postpone saves. For this reason, those files with current backups are considered first in computing unload eligibility. Note that a SAVE command can be included, prior to the UNLOAD command, if it is intended that all backups be immediately current, prior to unload eligibility factor determination.

The criterion should then be to select a set of files for unload which satisfies the request for space to ensure a maximum amount of time before any one of the files selected is referenced again.

SECURE defines a file's unload eligibility factor or 'UEF' as follows:

- The UEF can take on decimal values from 0 to 63.
- A UEF of 0 is reserved for files which are already unloaded. A UEF of 1 is reserved for catalogued tape files and removable disk files. A UEF of 2 is reserved for files which are marked 'unload inhibit' (V option). The higher the UEF, the greater is the chance that the file will be unloaded.

Factors considered in computing the UEF are:

1. Time of last reference.
2. Average time between references.
3. Size as given by the total number of granules.
4. Equipment type on which the file resides.
5. Do more recent F-cycles exist?
6. Is the file public or private?

All of these factors are included in the UEF formula.

9.9. OWN-PROJECT APPLICATIONS

The SECURE processor is used in privileged-mode runs to guarantee the security and availability of catalogued files. It is also available to non-privileged users for operations on their own files, or files to which they have the required access rights. Commands available to non-privileged users are: SAVE, LOAD, REVERT, LIST LAPSES and CLEAR LAPSES.

The only restrictions placed on the user in referencing SECURE commands from the previously described set are:

1. files named in source language namelists must be public or catalogued under the user's own project-id; and
2. both the read key and the write key, if they exist, must be included with the filename in source language statements.

The SAVE command, when referenced from within a non-privileged run produces a backup tape . The reel numbers however, are not recorded on the file's MFD item. This is necessary to maintain control over SECURE backup tapes created by a privileged-run, system-wide SAVE. With the modified or unrecorded SAVE, a user can produce backup tapes at will without destroying the record of the current backup tapes under the site manager's control.

The REVERT command may be called by the individual user after inadvertently destroying the text of the file on mass storage. If a backup copy exists, the SECURE processor marks the file as unloaded. An automatically initiated LOAD of the previous backup copy occurs when the file is next assigned. A non-privileged REVERT cannot be made on a read-only file. The user must first make the file read-write (@CHG) so that the REVERT process may release the file's granules.

The LIST LAPSES command is normally used by the individual user, following the assignment of a file, to see if any new lapses have occurred. The CLEAR LAPSES command may be used when the user is satisfied with the current state of the file and no longer cares to keep information about the file's previous history.

The LOAD commands and LOAD . . . FROM IBACKUP command may be used by the individual user to load any files, provided the complete filename and all required keys are specified in source language.

9.10. CATALOGUED FILE RECOVERY APPLICATIONS

No catalogued file should be considered disabled or destroyed as long as a valid SECURE-produced backup copy exists on tape. With this in mind, three modes of file recovery are possible under SECURE:

1. REVERT to the backup copy
2. REGISTER individual files from tape
3. REGISTER the directory tape

The REVERT command may be called by any user when it is necessary or desirable to make a tape backup copy become the primary copy. An automatically initiated LOAD of the designated backup copy's text to mass storage occurs when the file is next assigned.

Warning messages at assignment time signal when a file has been marked disabled. The user can then call @PRT,F (see 4.2.5) to get the status of the mass storage copy and time of backup creation. The Q option or an @ENABLE control statement (see 4.2.17) allows the user to probe the mass storage copy and determine whether a REVERT to a previous copy is required. Until the user makes the decision of which copy of the file to retain as the primary copy, the SECURE processor guarantees to preserve all the information it can to give the user the greatest possible choice of alternatives.

The most general mode of catalogued file recovery under SECURE involves using the MFD tape to recover all catalogued files, including catalogued tape files. To initiate this process, perform a REGISTER of the MFD tape. The SECURE processor copies this tape into a temporary mass storage file, catalogues all files not already catalogued, and marks all files except removable disk files and catalogued tape files as unloaded to backup tape. As a result, the system is restored to a condition at least as current as the time the MFD tape was created. This avoids doing REGISTERs of the separate backup tapes. As users attempt to assign files, an Executive function initiates an automatic LOAD of text back to mass storage. If it is possible that removable pack files were referenced since the time of MFD copy creation these packs should be removed and re-registered using the appropriate console operation keyins in order to insure directory consistency.

9.11. SUMMARY OF SECURE PROCESSOR COMMANDS

Table 9-2 lists the name and function of each SECURE processor command.

Table 9-2. Summary of SECURE Processor Commands

Command	Description
CLEAR LAPSES	This command erases existing memory lapse entries for the set of files specified by the namelist.
CONSOLE	This command designates the SECURE processor to accept source input images from the onsite console until an 'END' directive is received.
LIST FILES	Reserved for privileged runs, this command produces a summary listing, sorted alphabetically by qualifier and filename, of the backup status for all files specified in a namelist. If none is specified, all catalogued files are included.
LIST LAPSES	This command produces a listing of the memory lapse entries for all files or for the files specified by a namelist.
LIST REELS	Reserved for privileged runs, this command produces a summary listing of the current set of backups for all files in the specified namelist sorted in ascending reel number order.
LIST UNLOADED FILES	Reserved for privileged runs, this command produces the same summary listing as LIST FILES, but includes only those files whose text is currently unloaded.
LOAD	Reserved for privileged runs, this command causes the text of unloaded files given in a namelist to be retrieved from backup tape and written on mass storage.
LOAD...FROM IBACKUP	This command allows the user to REGISTER files (see REGISTER commands) and to LOAD their text from tape all in one operation.
REGISTER DIRECTORY FROM IBACKUP	Reserved for privileged runs, this command allows the user to REGISTER an entire system set of catalogued files using only the 'MFD tape' snapshot of the MFD.
REGISTER FROM IBACKUP	This command scans the set of backup tapes associated via source language with the particular IBACKUP unit and restores the MFD items of files found there which are not currently catalogued and whose complete backup copies reside on this reel set. Since only the directory items are restored with this command, files must be marked as unloaded to the backup tape.
REMOVE	This command is reserved for privileged runs and causes all files specified in a namelist to be deleted from the system after first ensuring that a current backup exists.
REMOVE ALL	This command is identical to the REMOVE command except that a new backup copy is produced for each file to be deleted, whether one already existed or not.
REVERT	This command causes a file's backup copy on tape to become the primary copy by releasing its granules on mass storage and marking the file as unloaded.
SAVE	Without further qualification, this command applies to all files in the system not catalogued with a G option if the run is privileged, or to only those files with a project-id matching that of the calling run if the run is not privileged. With the above noted, the SAVE command causes new backups to be made only for those files which do not have a current backup. Whether or not a backup is current depends on whether any write operations have been performed on the file since the time the last backup was made. In the case of privileged runs, the location of the new backup copy is recorded in the

Table 9-2. Summary of SECURE Processor Commands (continued)

Command	Description
	file's MFD items.
SAVE ALL	This is identical to the SAVE command except that a new backup copy is made regardless of whether a current backup exists. This allows the user to merge all backup copies on a new, self-contained set of backup tapes. In the case of an unloaded file, SAVE ALL retrieves the text of the file from tape instead of mass storage.
SAVE ALL WITH RECOVERY	This command is essentially the same as the SAVE ALL with the exception that files in the namelist specified are SAVEd only if their time of last backup creation is before the cataloguing time of SYS$*RECOVERY. (See 9.7.1)
UNLOAD	This command is reserved for privileged runs and without further qualification will not cause more files to be unloaded than is necessary to free up to 3000 tracks on mass storage. Files are chosen for unloading on the basis of their UEF (unload eligibility factor), which is computed by the SECURE processor. Files catalogued with a V option are not unloaded in any case. The UNLOAD command automatically implies SAVE; that is, a new backup copy is automatically produced, if required, before a file is marked as unloaded and its granules on mass storage are released.
UNLOAD TRACKS = *nnnnnn*	This command is identical to UNLOAD except that the preset limit of 3000 tracks is changed to the value specified.
UNLOAD POS = *nnnnnn*	This command allows the user to unload only position granular files up to only the number of positions specified.
UNLOAD specific files, projects, or accounts	This command differs from the previous unload commands in that all files, projects, or accounts specified in the namelist are unloaded regardless of the UEF.

9.12. EXAMPLES OF USE OF THE SECURE PROCESSOR

Example 1:

To make the set of backup tapes current (privileged):

```
@RUN
@ASG,A          SYS$*DLOC$/read key/write key
@ASG,TNF        OBACKUP,T
@SECURE,IL
 SAVE
END
```

Example 2:

To produce backups of user's own files (nonprivileged):

```
@RUN
@ASG,TNF      OBACKUP,T
@SECURE,ISF
 SAVE ALL FILES MY*FILE1, MY*FILE2
END
```

Example 3:

To merge all backup copies on a single set of tapes (privileged):

```
@RUN
@ASG,A        SYS$*DLOC$/read key/write key
@ASG,TN       IBACKUP,T
@ASG,TNF      OBACKUP,T
@SECURE,IL
 SAVE ALL
END
```

Example 4:

To revert to the backup copy (nonprivileged):

```
@RUN
@SECURE,ILF
 REVERT FILE ABC*XYZ
END
```

Example 5:

To load the text of certain files (privileged):

```
@RUN
@ASG,A        SYS$*DLOC$/read key/write key
@ASG,TN       IBACKUP,T
@SECURE,IL
 LOAD PROJECTS SATURN, JUPITER
 EXCEPT ACCOUNT 423055
END
```

Example 6:

To register a number of files from a particular backup tape set (nonprivileged):

```
@RUN
@ASG,TN       IBACKUP,T
@SECURE,IL
 IBACKUP = 1201, 1202, 1203
 REGISTER PROJECT MY-OWN FROM IBACKUP
END
```

Example 7:

To register an entire system set of files from the 'directory tape' following an initial boot (privileged):

```
@RUN
@ASG,A        SYS$*DLOC$/read key/write key
@ASG,TN       IBACKUP,T
@SECURE,IL
 IBACKUP = 1625G                                          . DIRECTORY TAPE
 REGISTER DIRECTORY TAPE FROM IBACKUP
END
```

Example 8:

To register and load the text for a number of files from a particular backup tape set (privileged):

```
@RUN
@ASG,A        SYS$*DLOC$/read key/write key
@ASG,TN       IBACKUP,T
@SECURE,IL
 IBACKUP = 1551, 1552, 1553
 LOAD FROM IBACKUP
END
```

9.13. MULTIPLE ACTIVITY OPERATION AND EXAMPLES

SECURE places all routines involving tape I/O in a reentrant ACTION segment which may be executed simultaneously by several activities. This is more efficient during massive, system-wide SAVE and LOAD runs.

Another essential aspect of optimizing multiple activity efficiency is the processor's ability to direct the allocation of files to specific mass storage units during the REGISTER and LOAD processes. The ability to direct allocation to an absolute subsystem and unit on a file-by-file basis is a feature of SECURE.

In describing the operational characteristics of multiple activities in SECURE, two points should be made clear at the outset. First, the number of activities generated by the processor for any SECURE action will be determined by the number of IBACKUP or OBACKUP tapes assigned to the calling run. SECURE can only do tape actions in one of two possible 'modes'. Any REGISTER or LOAD FROM IBACKUP command is defined as the 'register mode'. The other 'action' mode is signified by commands such as SAVE, UNLOAD, LOAD, REMOVE, etc. Therefore, the commands "REGISTER FROM IBACKUP1" and "SAVE TO OBACKUP1" are incompatible in the same execution.

A series of examples to illustrate proper use of source language in generating multiple activities is given below. All runs are assumed to be privileged.

Example 1:

```
@RUN
@ASG,TNF      OBACKUP1,T/2
@ASG,TNF      OBACKUP2,T
@SECURE,IS
 SAVE
END
```

In example 1, the processor uses an internal algorithm to generate two activities, each handling a unique file set and producing backup tapes of approximately equal size. The directory tape will be produced by the first activity, after both have terminated the SAVE action.

The assignment of OBACKUP1 illustrates the use of the two-unit tape assignment capability. By utilizing two tape drives, the SAVE operation continues even though a complete OBACKUP1 tape may have just been written. In this case, a second tape is mounted on an alternate drive and may be accessed while the first is rewinding.

Example 2:

```
@RUN
@ASG,TN       IBACKUP1,T
@ASG,TNF      OBACKUP1,T
@ASG,TNF      OBACKUP2,T
@SECURE,IS
 SAVE ALL FROM UNITS 4/0,5/0 TO OBACKUP1
 SAVE ALL FROM UNITS 6/0/1/2 TO OBACKUP2
END
```

In example 2, SECURE is directed to allocate files between the two activities based on their hardware location, because it has been determined that such a split offers the most efficient use of available I/O paths. Files which reside in part or both sets of mass storage units will be resolved and allocated to a single activity, normally on the basis of where the main directory item resides. All loaded files will be dumped first from drum to tape and then tape-to-tape copies of any unloaded files will be handled by a single activity using the IBACKUP1 tape unit for input and one OBACKUP tape unit for output.

Example 3:

```
@RUN
@ASG,TN       IBACKUP1,T
@ASG,TNF      OBACKUP1,T
@ASG,TNF      OBACKUP2,T
@ASG,TNF      OBACKUP3,T
@SECURE,IS
 SAVE ALL FROM UNITS 4/0,5/0 TO OBACKUP1
 SAVE ALL FROM UNITS 6/0/1/2 TO OBACKUP2
 SAVE ALL UNLOADED FILES TO OBACKUP3
END
```

Example 3 is similar to example 2, except that SECURE is directed to process unloaded files via tape-to-tape copies simultaneously under control of a third activity. Again, IBACKUP1 is automatically used as the input tape unit.

Example 4:

```
@RUN
@ASG,TN        IBACKUP1,T
@ASG,TN        IBACKUP2,T
@SECURE,IS
 IBACKUP1 =1,2,3
 IBACKUP2 =4,5,6
 LOAD FROM IBACKUP1
 LOAD FROM IBACKUP2
END
```

In example 4, SECURE is in 'register mode' and is directed to generate two activities to accomplish the REGISTER and LOAD. Allocation of the files on mass storage is done according to the normal system algorithm. The above example is suggested as the most efficient means to ensure proper allocation when restoring mass storage files.

Example 5:

```
@RUN
@ASG,TN        IBACKUP1,T
@ASG,TN        IBACKUP2,T
@SECURE,IS
 LOAD UNTIL UEF = 40
END
```

Following a directory tape REGISTER, the above method may be used to load file text up to the cutoff UEF value in a fast, efficient manner. Input tape numbers are known to SECURE through the directory items and are distributed evenly between the two activities, assuming approximately the same amount of information on each backup tape.

Although the above examples illustrate no more than three activities in one execution, as many as 10 will be allowed. However, the user should note that 4000 decimal words of storage are required for each additional activity and that there is a point of diminishing returns in generating too many activities for optimum I/O path selection. Note also that all the logic to successfully handle I/O errors and contingency processing has been preserved for each activity. A serious contingency occurrence, e.g., IGDM, causes the processor to do an ER ABORT$, terminating all activities, and then allows processing of the contingency.

9.14. SPECIAL FEATURES AND PROCEDURES

9.14.1. Checksum

The checksum feature is a means the SECURE processor uses to verify that data transfer in I/O operations is completed successfully. For each block of data transferred from mass storage to tape, tape to mass storage or tape to tape operations, a summation of the total bits is kept prior to and after the data transfer. If these summations are not equal, a checksum error has occurred.

These messages appear when this condition exists:

qual*file(cycle) ** CHECKSUM ERROR** nnnnnnnnnnnn nnnnnnnnnnnn

This line appears in the output listing once per track when an error is detected. The values printed after the message are, first, the actual checksum value after the transfer and second, the expected checksum prior to the I/O operation.

CHKSM ERROR qual*file(cycle) DISABLED

appears on the onsite console once per file when a checksum error has occurred. The file is left in the disabled state after SECURE has finished its operation.

This feature is included any time the C option is used on the SECURE control statement.

9.14.2. Text Block Sequence Check

The SECURE processor verifies that each text block of a file is transferred from tape by checking that the relative block numbers of all text blocks read are in sequential order. If a text block is not read, the user is notified of an error by the following messages:

qual*file(cycle) ** TEXT BLOCK SEQUENCE ERROR** nnn nnn

This line appears in the output listing each time a sequence error is detected. The values printed after the message are, first, the text block number of the previous block read and, second, the text block number of the last block read. Comparison of the two values enables quick determination of the number of blocks missed.

BLK SEQ ERR qual*file(cycle) DISABLED

appears on the onsite console once per file in which a sequence error occurs, and notifies the user that the file is left in the disabled state after SECURE has finished its operation.

9.14.3. 'Special Void' Message

When no projects, accounts, qualifiers or filenames are specified in a privileged mode REGISTER, the words 'SPECIAL VOID' are inserted as a dummy project name to indicate that the reference applies to all files, regardless of project.

This message appears in the summary listing if no legitimate files could be found on the tape or removable disk pack being registered.

9.14.4. Tape Handling Procedures

SECURE maintains its position on tape by checking the current file position number which is written in the file label block. If loss of position is suspected, a B keyin to an onsite console message will cause the tape to be rewound and the file search to be repeated from the beginning of the tape. A D keyin or the unsolicited downing of a tape unit will cause the particular activity to terminate via ER ERR$ without affecting other activities.

If a tape is mounted whose reel number in SECURE's tape label is not equal to the requested, SECURE will allow various actions by doing an answerable ER COM$. Based on the response, SECURE will react as follows:

A – Repeat the mount message as the wrong tape was mounted in error.

E – The wrong tape was mounted because the requested tape is currently unavailable. The files on this tape will be bypassed for this SECURE execution.

G – This is the requested tape, but the tape label has been destroyed. SECURE will proceed as if it had been correct, and attempt to recover the desired files.

9.14.5. SYS$*ARCHIVE$

SYS$*ARCHIVE$ is a V option file catalogued and used by SECURE as a storage area for selected directory information of files decatalogued on a REMOVE operation if the Z option is specified on the @SECURE control statement. By utilizing ARCHIVE$, a site may decatalogue seldom-used files from the Master File Directory, but still retain enough information to restore these files if the need arises. It must be assumed that files entered into ARCHIVE$ have a valid tape backup copy from which the text may be recovered.

Any namelist or limiter valid with the REMOVE or REMOVE ALL commands may be used to designate the set of files to be entered into ARCHIVE$.

A series of examples follow to illustrate the proper use of source language in conjunction with the creation, interrogation and purge of the file ARCHIVE$. All runs accessing ARCHIVE$ must be privileged.

Example 1:

To create entries in ARCHIVE$ for files with last reference times older than 30 days ago, and insure that these files are on a consolidated backup tape set:

```
@RUN
@ASG,A        SYS$*DLOC$/read key/write key
@ASG,TN       IBACKUP,U
@ASG,TNF      OBACKUP,U
@SECURE,ILZ
 REMOVE ALL BEFORE 30 DAYS
END
```

Example 2:

To create entries in ARCHIVE$ for a particular project:

```
@RUN
@ASG,A        SYS$*DLOC$/read key/write key
@ASG,TNF      OBACKUP,U
@SECURE,ILZ
 REMOVE PROJECT MYPROJ
END
```

Example 3:

To recreate a file with an entry in ARCHIVE$:

```
@RUN
@ASG,A          SYS$*DLOC$/read key/write key
@ASG,TN         IBACKUP,U
@SECURE,IL
 LOAD ARCHIVE FILE MYFILE FROM IBACKUP
END
```

Note in the above example that no IBACKUP = NNN tape number source statement is necessary. SECURE scans ARCHIVE$ and retrieves MYFILE's tape number automatically.

Because a file is deleted from the MFD when it is entered into ARCHIVE$, several entries with the same qualifier, filename and f-cycle may exist in ARCHIVE$. To insure recreation of the desired file, an additional specification, namely the time of cataloguing, may be stated in SECURE source language. Time of cataloguing may be obtained while the file is still in the MFD by many of the FURPUR @PRT commands (see 4.2.5). It must be stated in the format: 'CATALOGUED MM/DD/YY/HH/MM/SS' with leading zeros removed.

Example 4:

To recreate MYFILE catalogued on 04/18/73 at a time of 08:36:43:

```
@RUN
@ASG,TN         IBACKUP,U
@SECURE,IL
 LOAD ARCHIVE FILE MYFILE CATALOGUED 4/18/73/8/36/42 FROM IBACKUP
END
```

Periodically, ARCHIVE$ may be purged if retention of entries for specific files is no longer necessary. Files may be purged from ARCHIVE$ by 1) specifying filename(s), 2) specifying qualifier(s), or 3) based on the time spanned since files have been entered in ARCHIVE$.

Example 5:

To remove a specific file from ARCHIVE$:

```
@RUN
@SECURE,IL
 REMOVE ARCHIVE FILE MYFILE
END
```

Note that the cataloguing time and date of MYFILE (see Example 3) may be used on the REMOVE operation as well as the LOAD FROM IBACKUP, to insure that the correct file is purged from ARCHIVE$.

Example 6:

To remove a specific qualifier from ARCHIVE$:

```
@RUN
@SECURE,IL
 REMOVE ARCHIVE QUALIFIER MYQUAL
END
```

Example 7:

To remove files from ARCHIVE$ which were entered more than 30 days ago:

```
@RUN
@SECURE,IL
 REMOVE ARCHIVE BEFORE 30 DAYS
END
```

10. Symbolic Stream Generator (SSG)

10.1. INTRODUCTION

The Symbolic Stream Generator (SSG) processor provides a programmer with a means to create and control a variety of symbolic streams. With the use of a program or skeleton written in the SYMSTREAM language (see 10.5), directions and models may be used to manipulate symbolic input and create symbolic output.

10.2. SSG INPUT AND OUTPUT

10.2.1. @SSG Control Statement

Format:

@SSG,options param-1,param-2,param-3,param-4,param-5,param-6,param-7,...,param-n

All parameters are optional. All input and output streams defined by the @SSG control statement are DATA files or symbolic elements in program files, unless otherwise stated. SSG accepts mixed ASCII-Fieldata input. The only required input is the skeleton specified either on the SSG call, on a file-id statement, or in the input runstream.

Parameters:

options — See Table 10-1.

param-1 — Specifies the source of the skeleton stream (may be ASCII, Fieldata, or mixed ASCII-Fieldata mode).

param-2 — Specifies the source of SGSs; may be one of two types of input: first type is a DATA file or a symbolic element specified, which contains SGSs; second type has the format:

filename./label

where the filename specified is a program file. SSG takes the element names from the program file and attaches the specified label to each, creating one SGS for each element present. See 10.3 for the format of these SSG created SGSs. Both types of SGS input may also be used after param-6.

param-3 Specifies the destination of the generated output stream (must be a DATA file if *BRKPT directive is used). See V and W options, Table 10-1.

param-4 Specifies the destination of the revised temporary stream. Images will be the same mode (ASCII, Fieldata, or mixed ASCII-Fieldata) as the temporary images they are derived from.

param-5 Specifies the destination of the revised skeleton stream. These images will be in the same mode (ASCII, Fieldata, or mixed ASCII-Fieldata) as the skeleton and skeleton corrections are on input.

param-6 Specifies the source of the corrections to be applied to the skeleton stream (may be ASCII, Fieldata, or mixed ASCII-Fieldata). The control character for the line corrections to the skeleton is a plus sign (+).

param-7 The first of a variable number of fields specifying inputs; the formats are as follows (must always start at param-7):

PCF/number,name-1,...,name-n	permanent corrections
TCF/number,name-1 ,..., name-n	primary temporary corrections
SGS/number,name-1 ,..., name-n	stream generation statements
tcf-set-name/number, name-1 ,..., name-n	secondary temporary corrections sent to the tcf-set defined.

number — Specifies the number of DATA files or symbolic elements to be supplied for the specified input (PCF, TCF, ...).

name-1 ,..., name-n — The DATA files or symbolic elements that are the source for the specified input.

This format may be repeated any number of times by placing a comma (,) between name-n and the next input type. The source input may be in ASCII, Fieldata, or mixed ASCII-Fieldata mode.

If the P or T options are present for PCF and TCF sets, respectively, number must always be 1 and name-1 must be a program file. (See 10.4.2 and 10.4.3, respectively).

Table 10-1. @SSG Control Statement Options

Option	Description
A	Continue SSG execution regardless of no find references.
B	Do not dynamically @ADD the generated output stream. Without the B option the generated output stream is automatically added by the EXEC after SSG termination. (See 10.2.4 SSG OUTPUT).
C	Double space all printing.
D	Used with E option to eliminate indentation and preliminary error checking.

Table 10-1. @SSG Control Statement Options (continued)

Option	Description
E	Prints revised skeleton stream in an indented form and causes SSG to do preliminary error checking on the revised skeleton (see D option).
F	Prints permanent input streams (PCF set).
G	Prints temporary input streams (all TCF sets).
H	Prints revised temporary stream.
I	Prints stream generation statement streams in the order they are input.
J	Prints only the contents of the PCF element entries for which there are primary temporary element entries at the conclusion of skeleton processing. (The presence of *element/version in the primary TCF is sufficient to cause a printout of that element's PCF.) The page is ejected for each element printed out. The P option must be used. The R option is not necessary when the J option is used.
K	Prints generated output stream.
M	Debug. Prints trace of images in the skeleton as they are processed.
N	Debug. Prints values of variables and expressions of the directives being processed.
O	Order PCF set element entries in ascending alphabetical order. Any duplicate PCF element entries will be noted and the second entry discarded. (Sorted according to ASCII collating sequence.)
P	PCF set entries are to be composed of the symbolic element entries from a program file specified after param-6 on the @SSG call. When PCF entries are defined in this manner, other sources of PCF entries (except *CREATE PERM) are not allowed. (See 10.4.2)
Q	Order entries within each TCF set in ascending alphabetic order. Any duplicate entries within a TCF set will be noted and the second entry discarded. (Sorted according to ASCII collating sequence.)
R	Print updated or revised PCF set entries from the specified program file as they exist at the end of skeleton processing. A program file must have been used in conjunction with the P option.
S	Order SGSs with the same label in ascending alphabetical order according to the first subfield of each statement. (Sorted according to the ASCII collating sequence.)
T	TCF set entries are to be composed of the element entries from a program file specified after param-6 on the @SSG call (one program file per TCF set). When TCF entries are defined in this manner, other sources of TCF entries (except *CREATE TEMP) are not allowed. (See 10.4.3)
V	Generated output stream is to be Fieldata.*
W	Generated output stream is to be ASCII.*
X	Used in conjunction with any combination of the O, Q or S options; causes the sort to collate numbers in ascending order.

* If neither the V nor W option is used, the generated output stream is in ASCII if at least one revised skeleton image is in ASCII; otherwise, it is in Fieldata.

10.2.2. Input from Runstream

Another means of input to SSG is from the runstream (card input) where the file identification (file-id) statements define the type of input.

Format:

File-id Statement	Type of Input
SGS,options filename	stream generation statements
SKEL,options filename	skeleton stream
SKEL COR,options filename	corrections to skeleton stream
PERM COR,options filename	permanent corrections
TEMP COR,options filename	primary temporary corrections
TEMP COR,options filename : tcf-set-name	secondary temporary corrections sent to the tcf-set specified

Parameters:

options — W indicates that the card input should be read by ER AREAD$ (ASCII read). If file input, all images read are converted to ASCII. If W option is absent, card input is assumed Fieldata and all images in file input remain their original mode.

filename — The name of a data file or symbolic element that is the source for the specified type of input. Element notation is used to interpret the specification (a filename without an element must have a trailing period). The filename is not necessary, and if omitted, card input following the file-id statement is assumed.

Description:

If the filename is specified on the file-id statement, the images for the specified type of input are taken from that file. If the filename is omitted, SSG reads all card images following the file-id statement until an @EOF control statement is encountered and associates those images with the type of input specified. A matching @EOF is required for each file-id statement that uses cards as input.

Any number of file-id statements are allowed and may be in any order. A space period space on a file-id statement terminates the scan of that image. Space colon space on a TEMP COR file-id statement indicates that a tcf-set-name follows.

With two exceptions, any combination of input from the @SSG control statement and the file-id statement is permitted. (See P and T options in Table 10-1 for exceptions.)

10.2.3. SSG Input

The SSG skeleton is a collection of SYMTREAM statements interpreted by SSG in order to generate a symbolic output stream. Skeleton corrections are line corrections to be applied to the input skeleton before SSG interprets it. The control character for the skeleton corrections is always a plus (+).

The skeleton and skeleton corrections (if any) may be supplied by a parameter on the @SSG call, on a file–id statement, or as cards following a file–id statement. However, only one skeleton source and one skeleton correction source is allowed. Only the skeleton is required as input for an SSG call.

See 10.3 for description of SGSs, and 10.4 for description of PCF and TCF sets.

10.2.4. SSG Output

All output is optional. The generated output stream is symbolic images output as a direct result of the execution of the skeleton. With the skeleton, symbolic input and created symbolic images can be directed to the generated output stream.

The revised skeleton is the result of the input skeleton corrections being applied to the input skeleton stream. If there are no skeleton corrections, the revised skeleton is exactly like the input skeleton.

The generated output stream is automatically added by SSG before termination (for EXEC execution) unless the B option is on the @SSG call statement. SSG does an @ADD of the DATA file SGR$2 (containing the generated output stream) unless there is a user specified output file in param–3 on the @SSG call statement or on the *BRKPT directive that is the intended destination of the user output file to be added. In such a case the user's file is added and file SGR$2 is freed.

The revised temporary stream is discussed in 10.4.5.

The revised PCF set entries are those symbolic elements in the PCF program file (P option on @SSG call must be used – see 10.4.2) after a skeleton execution where *CORRECT,P was used. The mode (ASCII or Fieldata) of the correction images is not altered from input.

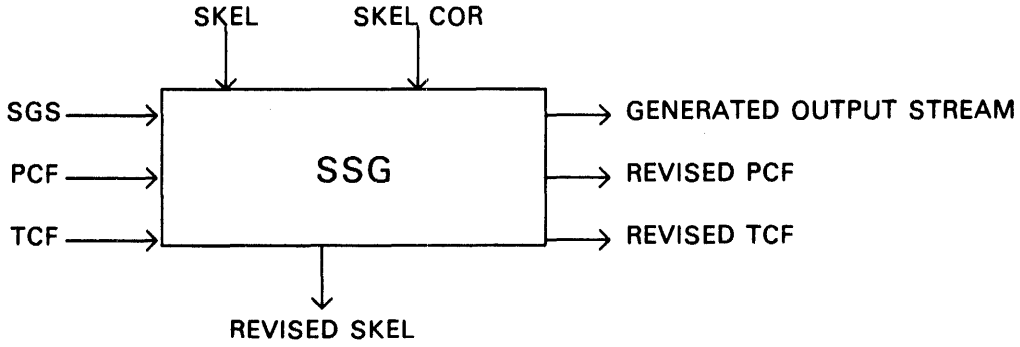


Diagram of SSG Input and Output

10.2.5. SSG Margins and Headings

After SSG prints the identification line, the output margins are reset. The user may determine the page margins by inputting a margin card with the following format:

MARGIN l,t,b

immediately after the @SSG call statement. l is the number of lines per page, t is the number of blank lines for the top margin, and b is the number of blank lines for the bottom margin. If there is no margin card present, SSG assumes 66, 6, and 3 for l,t, and b, respectively.

All of SSGs headings are printed by ER APRINT$ to ensure that any user heading (@HDG) is retained.

10.3. STREAM GENERATION STATEMENTS

Tables or lists of data may be provided for skeleton use through stream generation statements (SGS). An SGS stream may be input from the processor call statement, input from the runstream (cards), or created during skeleton processing.

10.3.1. SGS Input Formats

Format:

label subf-11,subf-12,...subf-1n subf-21,subf-22,...subf-2n ... subf-m1,subf-m2,...,subf-mn

Parameters:

label — String of 20 or fewer characters beginning with an alphabetic and not containing a space, period, semicolon, comma, left bracket, slash, plus, minus, or right bracket.

subf — Subfield: a string of 20 or fewer characters not containing a space, period, semicolon, comma, left bracket, or slash unless those characters are enclosed in apostrophes. Any character, including a space, is allowed in a given string by enclosing that string in pairs of apostrophes ("...") which designate the string, excluding the apostrophes. Single apostrophes ('...') designate a string including the apostrophes.

Description:

Each SGS has one label and any number of fields

In an SGS, a set of subfields separated by commas is called a field. Each of these fields are separated by blanks. Each field may have from one to any number of subfields. A null subfield is a subfield that contains no characters. Any number of SGS images with the same label and any number of differently labeled SGSs may exist.

The SGS is free form and the label need not start in the first character position (column 1). A period in any position on the SGS terminates the scan of that image; a semicolon continues the scan to the first nonblank character of the next image. The scan of an SGS ignores all leading blanks, but interprets the first trailing blank as a field separator and the first trailing comma as a subfield separator.

For input and referencing, an SGS label may be in upper or lower case alphabetics. Label LAB is the same as label lab, Lab, etc. However, the subfields in an SGS are maintained as they are received on input (Fieldata always converts to upper case in ASCII). This condition need not concern the Fieldata user.

When filename./label is supplied on the @SSG call (see 10.2.1, param-2), SSG automatically creates SGSs based on the nondeleted elements found in the specified program file's element table. The label specified becomes the label for the SSG created SGSs, and the format of those SGSs is as follows:

label element-name version-name element-type, element-subtype

If there is no version-name for an element, that field is left blank (12 blank characters are returned on reference or test to the field). If there is no element-subtype, that subfield is omitted. However, the element-name is always in field 1, subfield 1; the version-name is always in field 2, subfield 1; and the element-type is always in field 3, subfield 1.

10.3.2. Referencing SGSs

An SGS may be referenced in the skeleton according to label, statement number with that label, field, and subfield. When such a reference is encountered, the value returned is substituted for the reference. Those references below marked NUMERIC return a numeric value when SGS statement, field or subfield referenced exists, and in that case are considered numeric expressions.

With a label l, a statement number n, a field number f, and a subfield number s, the following references may be made on SGSs:

SGS Reference	Value Returned
[l]	Number of SGSs with the label l. (NUMERIC) If none, a 0 value is returned.
[l,n]	Number of fields on the n^{th} SGS with the label l. (NUMERIC) If the referenced statement does not exist, a 'no find' message is given.
[l,n,f]	Number of subfields in the f^{th} field on the n^{th} SGS with the label l. (NUMERIC) If the referenced statement or field does not exist, a 'no find' message is given.
[l,n,f,s]	Contents of the s^{th} subfield in the f^{th} field on the n^{th} SGS with the label l. (SYMBOLIC) If the referenced statement, field, or subfield does not exist, a 'no find' message is given.

In addition to the above references, information may be obtained about a particular subfield using the string descriptor numbers 1 through 5. All references are for the s^{th} subfield, in the f^{th} field on the n^{th} SGS with the label l. If the subfield referenced does not exist a 'no find' message is given.

SGS Reference	Value Returned
[l,n,f,s,1]	Number of characters in the subfield if every character is alphabetic; otherwise, 0. (NUMERIC)
[l,n,f,s,2]	Number of characters in the subfield if it is enclosed in single apostrophes ('...'); otherwise, 0. (NUMERIC)
[l,n,f,s,3]	Number of characters in the subfield, if it is enclosed in pairs of apostrophes ("...") otherwise, 0. (NUMERIC)
[l,n,f,s,4]	Number of characters in the subfield if it is numeric, otherwise, 0. (NUMERIC)
[l,n,f,s,5]	Number of characters in the subfield. (NUMERIC)

If SGSs are created by SSG as a result of filename./label being specified on the @SSG call, the external filename supplied is also saved and is associated with the specified label. In the case that more than one program file is associated with a label for this particular type of SGS input, any reference will return the last filename given with that label (from left to right on the @SSG statement). If a reference is made on a label that has no external filename associated with it, a 'no find' message is given. The following references may be made:

SGS Reference	Value Returned
[I,0,Q]	Qualifier associated with label; blanks, if none specified. (SYMBOLIC)
[I,0,F]	Filename associated with label I; blanks, if none specified. (SYMBOLIC)
[I,0,C]	F-cycle associated with the label I; blanks, if none specified. (SYMBOLIC)
[I,0,R]	Read key associated with the label I; blanks, if none specified. (SYMBOLIC)
[I,0,W]	Write key associated with the label I; blanks, if none specified. (SYMBOLIC)

See Volume 2-2.6.1 for explanation of file notation.

The fields in an SGS reference may be composed, as follows:

label	May be a string as described under label in 10.3.1, or an SGS reference, SET reference or a process parameter reference that returns such a string.
statement-number	May be an integer expression or numeric expression (see 10.5.1).
field-number	May be an integer expression or numeric expression (see 10.5.1).
subfield-number	May be an integer expression or numeric expression (see 10.5.1).
string descriptor number	May be an integer expression or numeric expression (see 10.5.1).
Q,F,C,R,W (on third field of external filename reference)	Explicit character only.

Any SGS references or set references used to supply the label, statement number, field number, subfield number, or string descriptor number may not contain another SGS reference or set reference. That is, SGS reference or set references may be nested to one level only.

10.3.3. SGS Examples

Example 1:

Given the following SGSs:

```
WHERE   183,ADOMAIN  MPLS MINN 56634
WHEN    21,OCTOBER,1954  +12465
WHO     CABLE,ABLE  WHERE,1  WHEN,2
WHERE   154,ROBERT  STPAUL  MINN  55127
WHO     BRAN,JULIET  WHERE,2  WHEN,1
WHEN    12,APRIL,1926  564Z
```

Also, given the variable Z (see 10.5.3.2) which has the value 1, and the process parameter references [#1] and [#2] which have the values WHO and 2, respectively (see 10.5.3.1.3).

The following references would return the values shown:

Reference	Value	Comment
[WHERE]	2	Number of SGSs with the label WHERE.
[[#1],1]	3	Number of fields on the first SGS with the label WHO.
[WHEN,Z,2]	1	Number of subfields in the second field of th first SGS with the label WHEN.
[WHO,2,1,[#2]]	JULIET	Contents of the second subfield in the first field of the second SGS with the label WHO.
[WHEN,1,2,[*Z],4]	0	Returns the number 0 because not all characters in the first subfield of the second field on the first SGS with the label WHEN are numeric.
[WHEN,[#2],2,1,5]	4	Number of characters in first subfield of second field on second SGS with the label WHEN.
[[WHO,Z,[#2],1],[[#1],Z,[#2],2] [*Z],[#2]]	ADOMAIN	Examples of how SGSs may be nested to one level. The label is taken from [WHO,Z, [#2],1] and is WHERE. The statement number is taken from [[#1],Z,[#2],2] and is 1. The field number is from the variable Z and is 1, and the subfield number is from [#2] and is 2.

Example 2:

Given the following SSG call and runstream:

```
@SSG    ,PF./NEWL,,,,,SGS/2,BETA/RD/WRIT./NEWL,ELL*DELTA(2)./OLDL
SKEL
.QUAL FOR NEWL IS [NEWL,0,Q].
FILENAME FOR NEWL IS [NEWL,0,F].
QUAL FOR OLDL IS [OLDL,0,Q].
F-CYCLE FOR OLDL IS [OLDL,0,C].
@EOF
@EOF
```

The generated output stream would look like:

```
QUAL FOR NEWL IS .
FILENAME FOR NEWL IS BETA.
QUAL FOR OLDL IS ELL.
F-CYCLE FOR OLDL IS 2.
```

1. Note that only the last program file associated with the label NEWL, from left to right on the call statement, can be referenced by the external filename reference.

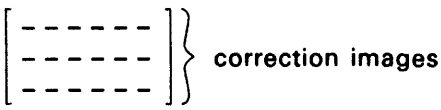
2. Note that when a reference was made on the qualifier associated with NEWL, and there was none specified, nothing was returned (blanks). (See 10.5.2 for bracketed references returning blanks.)

10.4. PERMANENT AND TEMPORARY STREAMS

In the EXEC 8 control language there is a facility for applying a group of correction images to a symbolic element and producing a new symbolic.

For example:

```
@FOR,U  FILE.SYMBOLIC
```



In the skeleton, there is often a need to direct or generate a symbolic stream (generated output stream) which contains correction images such as in the above example.

To faciliate handling the program corrections separately from the skeleton they are input as permanent and temporary streams (also called PCF and TCF, respectively). Then, by changing the permanent and temporary streams, groups of corrections can be changed each time a skeleton is used without actually changing the skeleton.

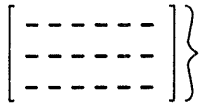
The permanent stream may contain many entries or groups of corrections to be applied to different symbolic elements. These groups of corrections are called element entries. The temporary stream, which is also made up of element entries, is provided as a means to input any necessary changes or additions to the permanent stream.

With the use of skeleton directives, permanent and temporary corrections may be merged and directed to the generated output stream (see 10.5.3.8).

10.4.1. Element Entry

An image made up of an asterisk (*) in the first character position followed by an element-name/version-name defines an element entry. All images that follow such an * card, until the next entry is defined, are attached to that element entry. The element name and version name may be up to 12 characters each, the characters being alphabetic, numeric, $ or –. The/version-name is optional, and the format of the entry is, as follows:

*element-name/version-name



Element-name/version-name are case independent (for ASCII users). That is, ABC/DEF is the same as abc/def. This condition need not concern the Fieldata user.

For alternate methods of defining element entries, see P and T options, Table 10-1, and below.

10.4.2. Permanent Stream (PCF)

The permanent stream or PCF is a set or collection of element entries. Images making up these entries may come from any combination of DATA files or symbolic elements specified on the @SSG call, on a file-id statement in the runstream, or cards from the runstream (see 10.2). Any number of element entries and the corresponding images may be in the PCF set as long as each entry name within the set is unique.

Another method for defining input to a PCF set is by specifying the P option (Table 10-1) along with a program file on the @SSG call (see 10.1, param-7). With this type of input, the name and contents of each element entry for the PCF set are taken from each nondeleted symbolic element in the one specified program file. With the use of the P option, no other type of input for the PCF set is accepted. Note that since the element entries are getting their names and contents from the symbolic elements in a program file, the * card should not be used to head the symbolic corrections within the element.

The PCF set is always defined, even when empty.

Example 1:

```
@SSG  ,,,,,,PCF/1,DATA.,PCF/2,PF.ELEA,DATA3.
PERM COR INT4
SKEL
     .
     .
     .
@EOF
PERM COR
*ENT2
     .
     .
     .
*ENT3/VER1
     .
     .
     .
@EOF
@EOF
```

Since the P option is not on the @SSG call many files and the run stream may be used to supply PCF set input. The DATA files, DATA.,DATA3., and INT4. and the symbolic element PF.ELEA each look something like the following:

```
*element/version
        .
        .
        .
*element/version
        .
        .
        .
*element/version
        .
        .
        .
```

Those files' element entries along with the element entries defined in the runstream between PERM COR and @EOF make up the PCF set.

Example 2:

```
@SSG,P  ,,,,,PCF/1,PF.
```

Because of the P option, the PCF element entry names are taken from the element table of the program file, PF. All nondeleted symbolic elements are included. If the file PF, had three nondeleted symbolic elements A,B, and C, there would be three PCF element entries, A,B, and C. The contents of the symbolic element PF.A would be the corrections associated with the element entry A; the contents of PF.B associated with B; and the contents of PF.C associated with C. Since the element entry names are defined from PF's element table, * cards should not be used in the correction symbolics.

10.4.3. Temporary Stream

A TCF set is a set or collection of element entries. Images making up these entries may come from any combination of DATA files or symbolic elements specified on the @SSG call, on a file-id statement in the runstream, or cards from the runstream (see 10.2).

Any number of element entries and the corresponding images may be in a particular TCF set as long as each entry name within that set is unique. SSG recognizes one primary TCF set and any number of secondary TCF sets. Common entry names between sets are permitted.

Input to be included in the primary TCF set is defined on the @SSG call as TCF (see 10.2.1, param-7) and on a file-id statement as:

```
TEMP COR filename

        or

TEMP COR filename : TCF
```

See 10.2 for further description.

Secondary TCF sets must be defined on the @SSG call by their set name or on a file-id statement, as follows:

TEMP COR filename : tcf-set-name

See 10.2 for further description.

A tcf-set-name is a maximum of six alphabetic characters, and the first three characters must be TCF (the characters TCF with no additional characters refers to the primary TCF set).

A secondary TCF set name must be defined in at least one of the two methods described above in order to be used in the skeleton. The primary TCF set is always defined (even when empty).

The purpose of having many TCF sets is to allow temporary corrections to be input from more than one source with the ability to merge them all together or just merge certain select groups of them. See 10.5.3.8.2 for further explanation.

Another means of defining a TCF set is by specifying the T option (see Table 10-1) on the @SSG call. Under this condition one program file may be supplied for each define TCF set. The name and contents of each element entry in a particular TCF set is taken from each nondeleted symbolic element in the program file specified for that set. With the T option, each TCF set must have its input supplied by a program file, and no other type of temporary input is accepted. Note that since the element entries are getting their names and contents from the symbolic elements in a program file, the * card should not be used to head the symbolic corrections.

Example 1:

```
@SSG ,,,,,,TCFA/1,DATAX.,TCFONE/2,PF1.EA,PF2.EB
TEMP COR INTT.
TEMP COR INTB. : TCFONE
TEMP COR : TCFX
*ENT1/VER1
     .
     .
     .
*ENT4/VER$
     .
     .
     .
@EOF
SKEL
     .
     .
     .
@EOF
@EOF
```

The element entries for the TCFA set are taken from the DATA file, DATAX. The element entries for the TCFONE set are taken from the symbolic elements PF1.EA,PF2.EB and the DATA file, INTB. The element entries for the primary TCF set are taken from the DATA file, INTT. The contents of the data files and symbolic elements mentioned above look something like the following:

```
*element/version
      .
      .
      .
*element/version
      .
      .
      .
*element/version
      .
      .
      .
```

The element entries for the TCFX set are taken from the images in the runstream between TEMP COR : TCFX and the next @EOF.

Example 2:

```
@SSG,T ,,,,,TCFR/1,PFX.,TCF/1,PFP.,TCFL/1,PF.
```

Because of the T option, the TCF sets defined must each have one program file specified. All TCF set's element entries are taken from the element table of the program file specified for that set. All nondeleted symbolic elements are included in a set. The symbolic elements in PFX. and their contents are the element entries in the secondary set TCFR. The symbolic elements in PFP. and their contents are the element entries in the primary set TCF. The symbolic elements in PF. and their contents are the element entries in the secondary set TCFL. The presence of the T option restricts all TCF set input to program files only.

10.4.4. Set References

A special mechanism, similar to the SGS referencing mechanism (see 10.3.2), may be used to access element entries. SSG has reserved certain labels for this purpose (these labels may not be used for SGS labels). The reserved label P is used for the PCF set, the reserved labels T or TCF are used for the primary TCF set, and each defined tcf-set-name (specified on the @SSG call or on a runstream file-id statement) is used as the reserved label for that set. The allowed references are as follows (those references below marked NUMERIC return numeric values when the entries referenced exist and in that case are considered numeric expressions):

Reference	Value Returned
[reserved-label]	Number of element entries included in the set with this reserved label. (NUMERIC) If a referenced set is empty or not defined, the value returned is 0.
[reserved-label,n]	1, if the n^{th} element in the set with this reserved label has only an element name; 2, if it has both an element name and a version name. (NUMERIC) If the referenced set is not defined, or there is not an n^{th} entry in the set, a 'no find' message is returned.

[reserved-label,n,f]	0, without exception. (NUMERIC) If the referenced set is not defined or there is not an n^{th} entry in the set, a 'no find' message is given.
[reserved-label,n,1,1]	Element name for the n^{th} element entry in the set with this reserved label. (SYMBOLIC) If the set referenced is not defined or there is no n^{th} entry in the set, a 'no find' message is given.
[reserved-label,n,2,1]	Version name for the n^{th} element entry in the set with this reserved label. (SYMBOLIC) If the set referenced is not defined or there is no n^{th} entry in the set, a 'no find' message is given.

The fields on a set reference may be supplied, as follows:

reserved-label	may be a string or an SGS reference, a process parameter reference or set reference that returns such a string.
rest of the fields	may be integer or numeric expressions.

Any SGS references or set references used to supply fields on a set reference may not contain another SGS reference or set reference.

The order of the element entries in a PCF or TCF set, unless controlled by the O and Q options, respectively, is determined by the order they are encountered on input. SSG reads across the SSG call statement, from left to right, and then down the runstream.

10.4.5. Revised Temporary Stream

A revised temporary element entry is a copy of an updated primary temporary element entry that could later be applied against an updated symbolic element, provided that the updated symbolic element was corrected by the same permanent element entry as was used to create the revised temporary element entry. That is, when a PCF set entry is applied to a symbolic element creating a revised symbolic element, a primary TCF entry whose line numbers were based on the old symbolic element could be revised to have its line numbers based on that revised symbolic element.

When a PCF entry and a primary TCF entry are merged via a *CORRECT (see 10.5.3.8.1) a revised temporary stream is generated.

A copy of the revised temporary stream is saved if the name of a data file or symbolic element is specified in param-4 of the @SSG call. A printing of the revised temporary stream is generated if the H option is on the @SSG call.

The revised temporary corrections will be in the same mode (ASCII or Fieldata) as the primary TCF stream they are derived from.

Example:

Primary TCF Entry:	PCF Entry:
-6,9	-1,2
-12,12	-10,10

A revised temporary would look like:

```
-4,7
-9,9
```

If the PCF entry were applied to some symbolic element, lines 1 and 2 and 10 would be removed from that element. Then, what was originally lines 6 through 9 are now lines 4 through 7 and what was line 12 is now line 9. The revised temporary is created to reflect the new line numbers.

10.5. SKELETON AND SYMSTREAM

A call on the SSG processor causes the interpretive execution of a program (commonly called a skeleton) which is written in a language called SYMSTREAM.

10.5.1. SYMSTREAM Primitives

The following terms are used by SYMSTREAM:

- numeric characters — 0,1,2,3,4,5,6,7,8,9
- alphabetic characters — A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z (upper and lower case for ASCII)
- special characters — #,Δ,),&,$,*,(,%,:,?,!,\,',¤, /,@,[,],−,+,<,=,>,comma,.,≠ ,:
- character — alphabetic character, numeric character, special character, blank
- string — collection of characters
- number — −99999,−99998,...,−1,0,1,...,999999
- integer expression — string of numeric characters
a number (string of explicit numbers)
variable name
set reference that returns a variable name (string)
SGS reference that returns a string that is an explicit number or variable name
process parameter that returns explicit number or variable name
integer expression ± integer expression
- numeric expression — ± integer expression
bracketed variable reference
SGS references that return numeric values (see lists under 10.3.2)
set references that return numeric values (see lists under 10.4.4)
numeric expression ± numeric expression

Examples:

Expression	Type of Expression
12	Integer (string of numeric characters or explicit number)
VAR3	Integer, where VAR3 is a variable

[XX,1,2,6]	Integer, where string returned is a variable name or explicit number
[#6]	Integer, where string returned is a variable name or explicit number
-146	Integer (explicit number) or numeric expression
[LABL]	Numeric expression (numeric SGS reference)
[*VAR3]	Numeric expression, where VAR3 is a variable
[TCF]	Numeric expression (numeric set reference)
A+B-[#2]	Integer expression, where A and B are variables and where [#2] returns an explicit number or variable name
+[R,2,6,1]-C	Numeric expression, where SGS reference returns explicit number or variable name and C is a variable

10.5.2. Nondirective Images

Any image in a skeleton that does not have an asterisk in the first character position (column 1) is a nondirective image. All nondirective images encountered in a skeleton are sent to the generated output stream, which may be printed later (K option, Table 10-1); or dynamically @ADDed by the EXEC after SSG terminates (B option, Table 10-1); or is output to a file whose filename is specified in Param-3 on the @SSG call (see 10.2.1).

Nondirective images may be created using any combination of characters (or strings), SGS references, process parameter references, variable references, and set references. The above listed references are distinguished from simple characters or strings (that are intended to go to the generated output stream directly as they are) by the brackets around them. All references are satisfied before SSG outputs the image. If a string returned from a bracketed reference is blanks (that are not enclosed in pairs of apostrophes), those blanks are not present in the output image.

Example:

Assume that the variable X has been defined and has the value 10, the process parameter [#1] has the symbolic value AUGUST, and the 1st PCF set element entry is AAA. Also assume that the following SGS is given as input:

```
ALPHA LEVEL 9 IN JUNE
```

Nondirective images in a skeleton, such as:

```
THE OLD LEVEL OF [P,1,1,1] FOR [ALPHA,1,4,1] WAS [ALPHA,1,2,1].

WE HOPE TO REACH LEVEL [*X] BY [#1].
```

would be placed in the desired output stream as:

```
THE OLD LEVEL OF AAA FOR JUNE WAS 9.

WE HOPE TO REACH LEVEL 10 BY AUGUST.
```

If a ¤ is explicit in the first character position of a nondirective image and the SSG reserved variable PILO$ is equal to zero, the ¤ is converted to an *. If PILO$ is nonzero, the conversion is not done.

If a # is explicit in the first character position of a nondirective image and the SSG reserved variable NSIGN$ is equal to zero, the # is converted to a @. If NSIGN$ is nonzero, the conversion is not done. PILO$ and NSIGN$ are initially set to zero and may be changed by the *SET or *CLEAR directives. (See 10.5.3.2.3 and 10.5.3.2.2, respectively). The ¤ is encoded in ASCII as " (double apostrophe).

Example:

If PILO$ is zero, the following nondirective image in the skeleton:

```
¤ THIS IS THE ANSWER*
```

would appear in the generated output stream as:

```
*THIS IS THE ANSWER*
```

If PILO$ is nonzero, the image would appear in the generated output stream exactly as it was in the skeleton.

If NSIGN$ is zero, the following nondirective image in the skeleton:

```
#ASG,T  TEMP,F
```

would appear in the generated output stream as:

```
@ASG,T  TEMP,F
```

If NSIGN$ is nonzero, the image would appear in the generated output stream exactly as it was in the skeleton.

Further means of manipulating nondirective images are discussed in 10.5.3.5 on *EDIT

10.5.3. Directive Images

The logical operations needed to determine if a nondirective image is to be sent to the generated output stream, or which nondirective images are to be sent, or in general, what paths will be taken in the skeleton, are performed by directive images. An asterisk in the first character position (column 1) of an image defines that image as a directive. The following are the symstream directives:

*BRKPT	*DIVIDE	*END	*MULTIPLY
*CLEAR	*DUMP	*IF	*PROCESS
*CORRECT	*EDIT	*INCREMENT	*REMOVE
*CREATE	*EJECT	*LOOP	*SET
*DEFINE	*ELSE	*MERGE	

Blanks may be placed on the image between the asterisk and directives for ease of readability. However, every image in the skeleton that has an asterisk in the first character position must be one of the above directives or have a period immediately following the asterisk (allows such images as comments). A period anywhere on a directive image causes SSG to terminate the scan of that image. (See 10.5.3.6.1, *BRKPT statement, for exception.)

The following conventions ae used in the syntax descriptions of the directives:

1. Δ signifies one or more mandatory blanks.
2. Items enclosed in [ ] brackets are optional (not to be confused with bracketed references).
3. Capitalized letters represent themselves and must be coded as shown.
4. Lower case parameters are filled in by the user.
5. Braces ( ), designate a choice of terms.

10.5.3.1. Defining Skeleton Image Sequences (Closed Subroutines)

The SYMSTREAM skeleton is broken down into two parts, the define section and the control section. The define section, which must be first in all skeletons, contains sequences of skeleton images that are placed in blocks between *DEFINE and *END directives. Each group of images with a define-end block is treated as a closed subroutine and stored for later reference. Any skeleton image encountered that is not between a matched *DEFINE and *END directive, defines the beginning of the control section. Once the control section begins no more define-end blocks are allowed. From the control section the define packets may be referenced (by *PROCESS) and provided with parameters. To reduce referencing time, the use of define packets should be reserved for larger groups of skeleton images. There is no limit on the number of define packets allowed.

10.5.3.1.1. *DEFINE - *END

Format:

*DEFINE Δ define-name

[- - - - - - - -]
[- - - - - - - -] skeleton images
[- - - - - - - -]

*END

Parameter:

define-name	20 or fewer characters supplied by a string, process parameter reference, SGS reference, or set reference

Description:

All images between the matched *DEFINE and *END directives are considered part of the define packet. None of the images in a define packet are executed until a call for that packet is made by the *PROCESS directive. From within a define packet, nested process calls on other define packets and recursive calls upon the same define packet are allowed to any depth.

10.5.3.1.2. *PROCESS

Format:

*PROCESS Δ define-name Δ [proc-para-1 Δ proc-para-2 Δ ... Δ proc-para-n Δ]

Parameters:

define-name — See 10.5.3.1.1 for description.

proc-para — Process-parameters are optional; 20 or fewer characters supplied by a string, process parameter reference, SGS reference, set reference, or numeric expression. If the string used contains a blank, period, semicolon, comma, left bracket, slash, plus, minus, or right bracket, and evaluation of that string is not intended, pairs of apostrophes should be used around it ("..."). For a string where the apostrophes are to be included, single apostrophes should be used ('...'). Double asterisks (**) can be used in front of an integer expression and it will be taken as a numeric expression to be evaluated. A single asterisk (*) followed by an SGS reference or process parameter reference directs the evaluation of the string that satisfies that reference.

Description:

The *PROCESS directive causes a define packet to be called for execution and passes parameters to that define packet. SSG examines all the parameters of the process directive, evaluates all numeric expressions and references made, and then converts any resulting numeric values to their symbolic representations. Therefore, only symbolic strings are passed to a define packet and any process parameter reference returns a symbolic string.

10.5.3.1.3. Process Parameter References

A reference to a process parameter within a define packet has one of the following formats (where n is some explicit number):

[#n] — Refers to the n^{th} process parameter specified on the last process call (which would be the process call on the define packet where this reference occurs).

[#n,define-name] — Refers to the n^{th} process parameter specified on the last process call for the given define-name (see 10.5.3.1.1 for description of define-name). Note that this reference must be in the define packet specified or in nested code (that is, this reference may be made in any define packet called by the define packet referenced). See Example 3.

All references return symbolic strings (not numeric values). See 10.5.3.1.2 under description.

Example 1:

With the following skeleton:

```
SKEL
*DEFINE ASM
#ASM,SU [ #1 ],[ #2 ]
*END
*PROCESS ASM ELTA ELTB
*PROCESS ASM ELTC ELTD
*PROCESS ASM ELTE ELTF
@EOF
```

The generated output stream would look like:

```
@ASM,SU ELTA,ELTB
@ASM,SU ELTC,ELTD
@ASM,SU ELTE,ELTF
```

Example 2:

With the following SGSs and skeleton:

```
    SGS
    INDIR " [LABL,1,1,1] ",Y
    @EOF
    SKEL
1.  *DEFINE SUBR
2.  [ #4 ][ #7 ][ #1 ] + [ #2 ] - [ #3 ][ #6 ][ #5 ].
3.  *END
4.  *DEFINE INIT
5.  *SET X TO 5
6.  *SET Y TO 6
7.  *SET Z TO 3
8.  *END
9.  *PROCESS INIT
10. *PROCESS SUBR [ *X ] **[INDIR,1,1,2] **Z *[INDIR,1,1,1];
11. +X+Y-Z IS [LABL,1,2,1]
    @EOF
```

The generated output stream looks like:

```
RESULT OF 5+6-3 IS 8.
```

1. Statements numbered 1 through 3 are part of the define packet SUBR.

2. Statements numbered 4 through 8 are part of the define packet INIT.

3. Statements numbered 9 through 11 are the control section.

4. Statement 9 causes the images in the INIT define packet to be executed. Thus, the global variables X, Y, and Z are defined.

5. Statements number 10 and 11 cause the image in the SUBR define packet to be executed and pass seven process parameters to it. The seven symbolic strings passed look like the following:

```
  5        after evaluation of numeric expression,[*X]
  6        after evaluation of **integer-expression, **[INDIR,1,1,2]
  3        after evaluation of **integer-expression, **Z
RESULT     after evaluation of *SGS-reference, *[INDIR,1,1,1]
  8        after evaluation of numeric expression, +X+Y-Z
  IS       string, IS
  OF       after return from SGS reference, [LABL,1,2,1]
```

   Note that the semicolon on statement 10 signals a continuation image.

6. Statement 2 has all the process parameter references satisfied and outputs the resulting image to the generated output stream.

Example 3:

With the following skeleton:

```
    SKEL
1.  *DEFINE LAST
2.  THIS IS LAST [#1,INIT] [#1,SECOND]
3.  *END
4.  *DEFINE INIT
5.  THIS IS INIT [#1]
6.  *PROCESS SECOND AXY [#1,INIT]
7.  *END
8.  *DEFINE SECOND
9.  THIS IS SECOND [#1],[#2]
10. *PROCESS LAST
11. END SECOND
12. *END
13. TEST ON PROC-PARA
14. *PROCESS INIT 2
15. DONE
    @EOF
```

The generated output stream would look like:

```
TEST ON PROC-PARA
THIS IS INIT2
THIS IS SECOND AXY,2
THIS IS LAST 2 AXY
END SECOND
DONE
```

1. Statement 13 (the first statement in the control section) is executed. Since it is nondirective, it is output to the desired output stream.

2. Statement 14 is executed and the define packet INIT is called and the parameter (2) is passed to it.

3. Statement 5 is executed (in the INIT packet) and 2 is substituted for [#1] before it is output.

4. Statement 6 is executed and the define packet SECOND is called and the parameters AXY and 2 (which is parameter 1 from the INIT call) are passed to SECOND.

5. Statement 9 is executed (in SECOND packet) and AXY and 2 are substituted for [#1] and [#2], respectively, since [#1] and [#2] refer to the last process call made. The image is output.

6. Statement 10 is executed and a call is made on the define packet LAST. No parameters are passed.

7. Statement 2 is executed (in LAST packet). The first parameter on the last call to INIT is referenced, which is 2 (call made at Statement 13). Also, the first parameter on the last call to SECOND is referenced, which is AXY (call made at Statement 6).

8. Since there are no more images in the LAST packet, SSG returns to the SECOND packet. Statement 11 is then executed (output). Since there are no more images in the SECOND packet, SSG returns to the INIT packet. There are no more images in the INIT packet so SSG returns to the control section of the skeleton and executes (outputs) Statement 15; SSG then terminates.

10.5.3.2. Symstream Variables

A variable name is a string of eight or less characters, not containing a space, period, semicolon, slash, plus, minus, left bracket, or right bracket.

Since SSG does its internal processing in ASCII, upper and lower case alphabetics are recognized. All references on variable names are, therefore, case dependent. The name XYZ is not the same as xyz. This condition need not concern the Fieldata user.

There are three ways in which a variable may be referenced (evaluated) in a directive image:

1. [*variable-name] . bracket reference

2. +variable-name . numeric reference

3. VALUE OF variable-name . only in *IF directives

In a nondirective image, only [*variable-name] is recognized as a variable reference.

Values of variables may be numeric only.

There are two types of variables described below, local and global. When a variable reference is made, the last created local variable of that name is searched for; if none exists, all global variables are searched for that name. Then, if none exists a no-find message is printed by SSG. In such a case, unless the A option is on the @SSG call (see Table 10-1), the processor terminates.

SSG has nine global variable names that are automatically defined when SSG is called and these variables are reserved for use by SSG and the user. The following are the reserved variables and a description of their use:

Reserved Variables	Description
CARD	When an *IF COLUMN SEARCH is true, CARD equals the statement number where the match was made (see 10.5.3.7.5).
FLD	When an *IF ROW SEARCH is true, FLD equals the field number where the match was made (see 10.5.3.7.5).

SFLD	When an *IF ROW SEARCH is true, SFLD equals the subfield number where the match was made (see 10.5.3.7.5).
MFLAG	Corresponds to M option on @SSG call; is zero when there is no M option, nonzero with the M option. By setting the value of MFLAG to zero or nonzero, the M option can be turned off or on, respectively, from the skeleton (see Table 10–1).
NFLAG	Corresponds to N option on @SSG call: is zero when there is no N option, nonzero with the N option. By setting the value of NFLAG to zero or nonzero, the N option can be turned off or on, respectively, from the skeleton (see Table 10–1).
PRTOFF	Initially set to zero (see 10.5.3.6.2)
PILO$	Initially set to zero (see 10.5.2).
NSIGN$	Initially set to zero (see 10.5.2).
COLED$	Is equal to the column number of the next character to be output when edit mode is on (see 10.5.3.5).

10.5.3.2.1. Skeleton Image Loops (Local Variables)

Purpose:

Local variables are created by the increment directive, and exist as long as incrementing is needed. Local variables may be created in this way even though a global variable with the same name exists.

Format:

INCREMENT Δ variable–name Δ [FROM Δ number] Δ [TO Δ number] Δ

[BY Δ number] Δ [WHILE Δ variable–name Δ IS Δ { SET / CLEAR }]

[----------]
[----------] skeleton images
[----------]

*LOOP

Parameters:

variable–name	See 10.5.3.2 for description; the variable–name may be supplied by a string, or an SGS reference, process parameter reference, or set reference that returns such a string.
number	Integer expression or numeric expression (see 10.5.1).

Description:

The FROM, TO, BY or WHILE phrases may be in any order, or absent. When absent, the FROM, TO, and BY values are assumed to be 1.

The increment directive creates a local variable of the name specified and initially sets its value equal to the FROM numeric (or 1). The first execution through the increment–loop packet is done if the FROM value has not exceeded the TO value limit, and the WHILE phrase is true, or absent. After the first execution through the increment–loop packet, the variable value is incremented by the BY numeric (or 1). Then, if the new variable value has not exceeded the TO value limit, and the WHILE phrase is true, or absent, the increment–loop is executed again. This increment–test process is repeated after every increment–loop execution. All images between the *INCREMENT and matching *LOOP are part of the increment–loop packet.

The variable value has exceeded the TO limit if:

((TO–value) – (present–value–of–variable)) X (BY – value) < 0

Note that with the above test, the BY value could be set negative and a decrementation process allowed. The WHILE phrase tests the specified variable (may be local or global) in that phrase to see if its value is zero (CLEAR) or nonzero (SET). If the test fails or the WHILE phrase is false, the increment–loop is over and the local variable created for that increment is destroyed. Loops may be nested to any level.

Example 1:

```
      SKEL
1.    *DEFINE X
2.    *INCREMENT A FROM -2 TO [#2] BY [BY,1,1,1]
3.    *INCREMENT [VAR2,1,1,1] FROM 0 TO -3 BY -2 WHILE [#3] IS SET
4.    *INCREMENT [#1] FROM 0
5.    *SET C TO [#1]
6.    [*A] [*B] [*C]
7.    *LOOP
8.    *LOOP
9.    *LOOP
10.   *END
11.   *PROCESS X VAR 0 A
      @EOF
      SGS
      VAR2 B
      BY 2
      @EOF
```

The generated ouput stream would look like:

```
-200
-201
-2--20
-2--21
```

10.5.3.2.2. Creating and Changing Global Variables (*CLEAR)

Format:

*CLEAR Δ variable-name

Parameter:

variable-name — See 10.5.3.2 for description; it may be supplied by a string, or SGS reference, process parameter reference, or set reference that returns such a string.

Description:

The value of the last local variable with the given name is set to zero. If no local variable of that name exists, the global variables are searched for one with that name, and it is set to zero. If no global variable is found, one with that name is created and given the value of zero. Once created, this global variable exists for the remainder of the skeleton processing.

Example:

```
    SKEL
1.  *DEFINE X
2.  [*A] INSIDE X
3.  *CLEAR [#1]
4.  *END
5.  *CLEAR [VARCLR,1,1,1]
6.  *INCREMENT A FROM -2 TO -1 BY 1
7.  *PROCESS X A
8.  [*A] INSIDE INC LOOP
9.  *LOOP
10. *CLEAR A
    @EOF
    SGS
    VARCLR PP
    @EOF
```

The generated output stream would look like:

```
-2 INSIDE X
0 INSIDE INC LOOP
```

1. Statement 5 creates a global variable called PP with the value 0 since no local or global variables of that name are found.

2. Statement 3 clears the variable A (passed as parameter one on the process call at statement 7). Since a local variable is found with that name it is set to 0 and no more searching is done. Since the *PROCESS call is within the increment loop, the define packet called is also in that loop and thus, the local variable A is defined in the define packet X.

3. Upon return from the define packet X, the local variable A is now 0 and fails the TO limit test so the increment loop is destroyed and the local variable A is destroyed also.

4. Statement 10 creates a global variable called A since no local or global variables of that name are found (local variable A was destroyed when the increment loop terminated).

10.5.3.2.3. Creating and Changing Global Variables (*SET)

Format:

*SET Δ variable-nameΔ [TO Δ number Δ]

Parameters:

variable-name	See 10.5.3.2 for description; may be supplied by a string, or SGS reference, process parameter reference, or set reference that returns such a string.
number	Integer expression or numeric expression (see 10.5.1).

Description:

If the TO phrase is omitted, TO value is assumed 1. The value of the last local variable with the specified name is set to the TO value. If no local variable with that name exists, a global variable with that name is searched for and set to the TO value. If no global variable with that name exists, one is created with the given TO value. Once created, this global variable exists for the remainder of skeleton processing.

Example:

Given the following SGSs and skeleton:

```
    SGS
    VARSET VAR2,8
    @EOF
    SKEL
1.  *DEFINE X
2.  [*A]
3.  *SET [#1] TO [#1] +2
4.  *END
5.  *SET A TO 6
6.  *INCREMENT A TO 3
7.  *PROCESS X A
8.  [*A]
9.  *LOOP
10. [*A]
11. *SET [VARSET,1,1,1] TO [VARSET,1,1,2]
    @EOF
```

The generated output stream would look like:

```
1
3
6
```

1. Statement 5 creates a global variable A with the value of 6 since no variable with that name already existed.

2. Statement 6 creates a local variable A with the initial value of 1.

3. The define packet X is called at Statement 7 and the string A is passed as parameter 1.

4. Statement 2 references the variable A and the value 1 is returned since the local variable A was found first.

5. Statement 3 does a *SET A to A + 2 since the string in [#1] was passed as A. SSG looks for a variable A (and finds a local variable of that name) and sets A equal to itself +2 which is 3.

6. Statement 8 references the variable A and the value 3 is returned since the local variable A is found first.

7. The increment loop terminates since A (equal to 3) + BY value (assumed 1) is greater than the TO value (which is 3).

8. Statement 10 references the variable A and returns the value 6 after finding the global variable A. The local variable A no longer exists since the increment loop is destroyed.

9. Statement 11 creates a variable VAR2 with the value of 8.

10.5.3.2.4. Variable Multiplication (*MULTIPLY)

Format:

*MULTIPLY Δ number Δ BY Δ number Δ GIVING Δ variable-name

Parameters:

variable-name	See 10.5.3.2 for description; may be supplied by a string, or SGS reference, process parameter reference, or set reference that returns such a string.
number	Integer expression or numeric expression (see 10.5.1)

Description:

The product of the first number multiplied by the second number is set as the value of the given variable-name. The variable specified must already exist (be defined) and may be either a local or global variable. In satisfying the search for the variable-name, the local variables are searched first (from most recently defined backwards), then the global variables are searched.

Example:

Given the following skeleton and SGSs:

```
     SKEL
1.   *DEFINE MULTP
2.   *MULTIPLY [#1] +5 BY [VARSUP,1,1,1] GIVING VAR3
3.   *END
4.   *CLEAR VAR3
5.   [*VAR3]
6.   *PROCESS MULTP 4
7.   [*VAR3]
     EOF
     SGS
     VARSUP 4
     @EOF
```

The generated output stream would look like:

```
0
36
```

1. Statement 4 creates the global variable VAR3 with the value of 0 and statement 5 when referencing that variable returns the value 0.

2. Statement 6 calls the define packet MULTP and passes the first parameter as 4.

3. Statement 2 multiplies [#1]+5, which is 9, by [VARSUP,1,1,1] which is 4 and sets the global variable VAR3 to the product, 36.

4. Statement 7 references the variable VAR3 and returns with the value 36.

10.5.3.2.5. Variable Division (*DIVIDE)

Format:

*DIVIDE Δ number Δ BY Δ number Δ GIVING Δ variable-name [,variable-name] Δ

Parameters:

number	Integer expression or numeric expression (see 10.5.1).
variable-name	See 10.5.3.2 for description; may be supplied by string, or SGS reference, process parameter reference, or set reference that returns such a string.

Description:

Divides the first number by the second number and sets the first variable specified equal to the quotient. The second variable-name specified is optional, and if present, is set equal to the value of the remainder. The variable-names specified must already exist (be defined) and may be either local or global variables. In satisfying the search for the variable-names, the local variables are searched first (from the most recently defined backwards), then the global variables are searched.

Example:

Given the following SGSs and skeleton:

```
    SGS
    CONST 3
    @EOF
    SKEL
1.  *DEFINE DIV
2.  *DIVIDE [#1] BY + [CONST,1,1,1] GIVING QUO,REM
3.  *END
4.  *CLEAR QUO
5.  *CLEAR REM
6.  *PROCESS DIV 8
7.  QUOTIENT IS [*QUO], REMAINDER IS [*REM]
8.  *DIVIDE REM BY 2 GIVING QUO
9.  [*REM] DIVIDED BY 2 IS [*QUO]
    @EOF
```

The generated output stream looks like:

```
QUOTIENT IS 2, REMAINDER IS 2
2 DIVIDED BY 2 IS 1
```

1. Statements 4 and 5 create QUO and REM as global variables with the value 0.

2. Statement 6 calls the define packet DIV and passes 8 as parameter 1.

3. Statement 2 divides [#1], which is 8, by +[CONST,1,1,1], which is 3 and sets QUO equal to the quotient of 2 and REM equal to the remainder 2.

4. Statement 7 is output with the values for QUO and REM substituted.

5. Statement 8 divides REM, which is equal to 2 by 2 and set QUO equal to the quotient of 1.

6. Statement 9 is output with the values for REM and QUO substituted.

10.5.3.2.6. Variable Dump (*DUMP)

Formats:

1. *DUMP Δ { LOCAL / GLOBAL } Δ

2. *DUMP Δ [{ LOCAL / GLOBAL } Δ] variable-name-1 [,...,variable-name-10]

Parameters:

variable-name	See 10.5.3.2 for description; may be supplied by a string or a process parameter reference that returns such a string.

The *DUMP directive is intended as a debug aid printing values of variables in the SSG run log.

*DUMP GLOBAL	Prints in SSGs run log values of all global variables (the global variable list includes the SSG reserved variables).
*DUMP LOCAL	Prints in SSGs run log the values of all local variables.
*DUMP LOCAL; variable-name-1,...,variable-name-10	Prints in SSGs run log the values of from one to ten specified local variables.
*DUMP GLOBAL; variable-name-1,...,variable-name-10	Prints in SSGs run log the values of from one to ten specified global variables.
*DUMP; variable-name-1,...,variable-name-10	Prints in SSGs run log the values of from one to ten variables searching first the local variables for the name specified, then the global variables.

If the variable reference is not found, the words NO FIND are substituted for the value of the variable in the run log print.

10.5.3.3. Internal Chains

SSG maintains three major internal chains; the PERM chain, the TEMP chain, and the SGS chain. They consist of the PCF set entries, the primary TCF set entries (each secondary TCF set also has a chain but it cannot be altered), and stream generation statements, respectively. Entries in these three chains can be created and removed internally (only for the duration of the SSG execution).

10.5.3.3.1. Dynamic Expansion of Internal Chains (*CREATE)

Formats:

1. *CREATE Δ SGS: Δ sgs-image Δ

2. *CREATE Δ { PERM: / TEMP: } Δ element-name [/version-name Δ]

Parameters:

sgs-image	See 10.3.1 for the construction of an SGS image. Any bracketed references may be used to supply the strings for the label or subfields on the first card image only (not on any continuation images).
element-name version-name	See 10.4.1 for allowable element entry names; the format may be element or element/version where the version name is optional. The element name or the version name may be supplied by a string, SGS reference, process parameter reference, or set reference.

Description:

1. Creates an SGS with the sgs-image after all references have been satisfied and adds it to the bottom of the SGS chain (unless the S option is used - see Table 10-1). In the SGS chain, SGSs are grouped according to label. Once created, the new SGS is treated and may be referenced like any other input SGS. An *CREATE SGS: image may be edited by use of the *EDIT on directive (see 10.5.3.5); however, the created SGS must not exceed 80 characters.

2. An element entry with the name element–name/version–name or just element– name is inserted at the bottom of the PERM (PCF set) or TEMP (primary TCF set) chain (unless the O or Q options, respectively, are used – see Table 10–1). Such created entries may be referenced via the set references and may be used when merging correction entries (treated as empty entries). However, an *IF test on correction entry existence (see 10.5.3.7.6) on a created entry always yields a Boolean False since no input streams are attached to it.

   Even though the CREATE image can be longer than one card image each (by use of the semicolon (;) and continuation images), bracketed references will be recognized on the first card image only.

Example:

Given the following SGSs and skeleton:

```
    SGS
    OLDSGS XYZ
    @EOF
    SKEL
1.  *DEFINE CRT
2.  *CREATE SGS: NEWSGS [#1],[OLDSGS,1,1,1] [OLDSGS,1,1,1,5],[*A]
3.  *END
4.  *SET A TO 40
5.  *PROCESS CRT PROCP
6.  [NEWSGS],[NEWSGS,1,1,1],[NEWSGS,1,2]
7.  *CREATE PERM:  ELEM1
8.  *CREATE TEMP:  ELEM2/[OLDSGS,1,1,1]
9.  [P,1,1,1] IS PCF ENTRY
10. [T,1,1,1]/[T,1,2,1] IS PRIMARY TCF ENTRY
11. *IF ELEM1 HAS CORRECTIONS
12. PCF CORR
13. *ELSE
14. NO PCF CORR
15. *END
    @EOF
```

The generated output stream looks like:

```
1,PROCP,2
ELEM1 is PCF ENTRY
ELEM2/XYZ IS PRIMARY TCF ENTRY
NO PCF CORR
```

1. The SGS created at Statement 2 and put in the SGS chain looks like:

   ```
   NEWSGS PROCP,XYZ 3,40
   ```

2. The entry ELEM1 is created in the PCF chain in Statement 7.

3. The entry ELEM2/XYZ is created in the primary TCF chain in Statement 8.

4. Output from statements 9 and 10 show that entries have been made on the PCF and PRIMARY TCF chains.

5. The *IF test on correction entry existense still shows no corrections as existing.

10.5.3.3.2. Deleting Entries from Internal Chains (*REMOVE)

Formats:

1. *REMOVE Δ SGS Δ sgs-label [,sgs-start-stmt [,number-to-remove]] Δ

2. *REMOVE Δ { PERM / TEMP } Δ element-name [/version-name]

Parameters:

sgs-label	Certain SGSs with this label are to be removed. See 10.3.1 for description of an SGS label; it may be supplied by a string, SGS reference, process parameter reference, or set reference that returns such a string.
sgs-start-stmt	The statement number with the given sgs-label where the remove will start; may be an integer expression or a numeric expression (see 10.5.1). This number is optional, and if omitted 1 is assumed.
number-to-remove	The number of SGSs to remove; may be an integer expression or a numeric expression (see 10.5.1). This number is optional, and if omitted, 1 is assumed.
element-name version-name	See description under 10.5.3.3.1

Description:

1. Removes a specified number (or 1) of SGSs with the specified label starting at the specified start number (or 1).

2. Removes the element entry with the specified name from the PERM (PCF) chain or TEMP (primary TCF) chain. Once removed, the entry and any corrections associated with it are destroyed during the SSG execution.

Example:

Given the following skeleton and SGSs:

```
      SKEL
1.    *DEFINE REM
2.    *REMOVE SGS [#1],2
3.    *REMOVE PERM [#2]/[VERS,1,1,1]
4.    *END
5.    *PROCESS REM LABL [ELE,1,1,1]
6.    *REMOVE TEMP A2
      @EOF
      SGS
      ELE A1
      VERS B1
      LABL X
      LABL Y
      LABL Z
      @EOF
```

1. Statement 2 removes one SGS with the label LABL starting at statement number 2.

2. Statement 3 removes the entry A1/B1 from the PCF chain.

3. Statement 6 removes the entry A2 from the primary TCF chain.

10.5.3.4. *EJECT

Format:

```
*EJECT
```

Description:

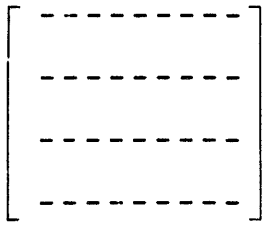
The *EJECT directive may appear anywhere in the revised skeleton and is transparent to any skeleton processing. When the revised skeleton is being printed using the E option without the D option, and this directive is encountered, a page eject is done.

If the D option is present with the E option, the *EJECT is ignored. This directive is intended to make the revised skeleton easier to read.

10.5.3.5. Concatenating Nondirective Images (*EDIT)

Format:

*EDIT Δ ON Δ [edit-symbol] Δ



*EDIT Δ OFF Δ

Parameters:

edit-symbol — One character which may be alphabetic, numeric, or special character (see 10.5.1), or an SGS reference which, when satisfied, is one of the above characters. The edit symbol is optional and when omitted, & (ampersand) is assumed.

Description:

Edit-mode may be used to build a non-directive or a *CREATE image from more than one symbolic image.

The edit-mode is turned on by the *EDIT ON directive. When the edit mode is on, all nondirective or *CREATE images which are terminated by the chosen edit symbol are joined together (concatenated) and output to the generated output stream. If a nondirective edit image that SSG is creating gets larger than a card image (80 characters), a semicolon is automatically inserted and the image is continued on the next line. An *CREATE edit image cannot exceed 80 characters.

If a nondirective is being edited any directive image may be executed while the edit mode is on but the occurrence of any nondirective image that does not have the chosen edit symbol on it, causes the edit mode to terminate. Otherwise, the edit mode is normally terminated by the *EDIT OFF directive, and the final created image is output.

If a *CREATE directive is being editted it must immediately follow the *EDIT ON directive. Any other directive image may be executed while edit mode is on but the occurrence of any portion of the *CREATE image that does not have the edit symbol on it causes the edit mode to terminate. Otherwise, the edit mode is normally terminated by the *EDIT OFF directive and the final created image added to the list of *CREATE images.

Further control of an edit image is provided by the reserved variable COLED$. Based on an 80 character image, COLED$ reflects the next character position to be written in the created edit image. If the edit mode is off, COLED$ is zero. Any variable directive may be used to control COLED$ and thus control the construction of an image while in edit mode. However, any change to that variable is not recognized until the next nondirective image is processed for editing.

Example 1:

Given the following SGSs and skeleton:

```
SGS
MONTH JUNE      30,42,15
MONTH JULY      24,7,94
MONTH AUGUST    1,171,463
TAB  10
TESTS 3
@EOF
SKEL
*EDIT ON
MONTH &
*CLEAR INDEX
*SET TAB TO [TAB,1,1,1]
*INCREMENT A TO [TESTS,1,1,1]
*MULTIPLY [*TAB] BY [*A] GIVING COLED$
TEST [*A] &
*LOOP
*EDIT OFF
*INCREMENT DATE TO [MONTH]
*EDIT ON !
[MONTH,DATE,1,1] !
*INCREMENT VAL TO [TESTS,1,1,1]
*MULTIPLY [*TAB] BY [*VAL] GIVING COLED$
[MONTH,DATE,2,VAL] !
*LOOP
*EDIT OFF
*LOOP
@EOF
```

The generated output stream would look like:

MONTH	TEST1	TEST2	TEST3
JUNE	30	42	15
JULY	24	7	94
AUGUST	1	171	463

Example 2:

Given the following skeleton:

```
SKEL
*EDIT ON !
*CREATE SGS : FILE [TAPE,1,1,1] !
[FAST,1,1,1]
*HDG **TABLE of CONTENTS **
@EOF
SGS
TAPE TEST1
FAST TEST2
@EOF
```

The following SGS would be created:

```
FILE TEST1  TEST2
```

10.5.3.6. Directing the Generated Output Stream

Normally, SSG maintains an internal SDF file (SGR$2) where all generated output images are sent. This can be suppressed by the use of the B option (Table 10–1) without specifying either the K option (Table 10–1) or a filename in parameter 3 of the @SSG call (see 10.2). By use of the *BRKPT directive, the generated output stream can be broken apart and sent to different SDF files. The K option can be used to control printing parts or all of the generated output. When the generated output stream is sent to user-specified files, it is copied from SGR$2 to the user file after it is created. This makes nested @SSG calls impossible since SGR$2 will only contain the latest output stream.

10.5.3.6.1. Breakpointing Images(*BRKPT)

Format:

*BRKPT,[options] Δ [filename Δ]

Parameters:

file-name — The name of a SDF file. The filename must be in element notation. Therefore, for this particular SSG directive statement, only a space period space will terminate the scan of the image (see 10.5.3). It specifies the destination of all generated output images after the *BRKPT until the next *BRKPT or the end of the skeleton. The filename may be supplied by a string, SGS reference, set reference or process parameter reference. If omitted, the SSG internal file is assumed.

options:

K — Print the generated output images following this *BRKPT until the next *BRKPT or the end of the skeleton.

Description:

All generated output images that are encountered before the first *BRKPT in a skeleton are directed by the K option (or the absence of it) on the @SSG call, and are sent to the file specified in parameter 3 of the @SSG call (if one was specified). The B option on the @SSG call controls only the last generated output stream. That is, only the generated output images that are encountered between the last *BRKPT executed in the skeleton and the end of the skeleton, are controlled by the B option. If the *BRKPT directive is not used in the skeleton, the B and K options control the entire generated output stream.

Example:

Given the following SSG call:

```
@SSG,K  ,,FILEA.
SKEL
IMAGE A1
IMAGE A2
*BRKPT FILEB.
IMAGE B1
IMAGE B2
*BRKPT,K  FILEC.
IMAGE C1
IMAGE C2
*BRKPT
#ASG,T  TEMP,F
#MSG,N  TEMP WAS ASSIGNED
@EOF
@EOF
```

The generated output stream printings would look like:

Generated output stream part 1

```
IMAGE A1
IMAGE A2
```

Generated output stream part 3

```
IMAGE C1
IMAGE C2
```

The part of the generated output stream that would be dynamically added by SSG is:

```
@ASG,T  TEMP,F
@MSG,N TEMP WAS ASSIGNED
```

The contents of FILEA would be

```
IMAGE A1
IMAGE A2
```

The contents of FILEB would be

```
IMAGE B1
IMAGE B2
```

The contents of FILEC would be

```
IMAGE C1
IMAGE C2
```

10.5.3.6.2. PRTOFF

The printout of the generated output stream can be dynamically controlled by the use of the SSG reserved variable PRTOFF. Initially, PRTOFF is set to zero. When PRTOFF is set to a nonzero value, the printout is suppressed. The *SET and *CLEAR directives may be used to control PRTOFF. The printout of generated output images may be turned on and off any number of times. Even though the print of generated output images is suppressed, the images are still in the generated output stream.

Example:

Given the following SSG call:

```
@SSG,K
SKEL
#ELT,I .ELEA
*SET PRTOFF TO 3
AA1
AA2
*CLEAR PRTOFF
#ELT,I .ELEB
*SET PRTOFF
BB1
BB2
#EOF
@EOF
@EOF
```

The print of the generated output stream would look like:

```
@ELT,I  .ELEA
@ELT,I  .ELEB
```

However, the generated output stream would actually be:

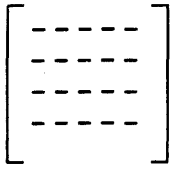
```
@ELT,I  .ELEA
AA1
AA2
@ELT,I  .ELEB
BB1
BB2
@EOF
```

and the immediately above would be dynamically added by SSG.

10.5.3.7. Skipping Skeleton Images (*IF),(*ELSE),(*END)

*IF is a decision making directive designed to conditionally test, and choose logical paths in the skeleton, by either executing or skipping a sequence of skeleton images. There are two general formats of IF packets, as follows:

*IF (Conditional)

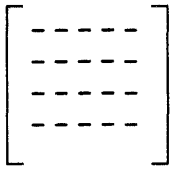


skeleton images

*END

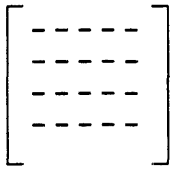
Where any directive or nondirective skeleton image (except *DEFINE) may be between the matching *IF and *END. If the condition is true, the skeleton images between the *IF and *END are executed; if the condition is false, they are skipped.

*IF (Conditional)



skeleton images

*ELSE



skeleton images

*END

Where any directive or nondirective skeleton image (except *DEFINE) may be between the matching *IF and *ELSE and the matching *ELSE and *END. If the condition is true, the skeleton images between the *IF and *ELSE are executed and the images between the *ELSE and *END are skipped. If the condition is false, the skeleton images between the *IF and *ELSE are skipped and the images between the *ELSE and *END are executed.

IF packets may be nested to any level, but no overlap of IF packets with other IF packets or with increment-loops is allowed. Each increment loop or IF packet must be entirely nested within another IF packet or increment-loop.

Example: allowed

*IF (Conditional)

- - - - - - - -
- - - - - - - -
- - - - - - - -
*ELSE

```
*INCREMENT A TO 3
|
|  - - - - - - - -
|  - - - - - - - -
|  - - - - - - - -
|  *LOOP

*IF (Conditional)
|
|  - - - - - - - -
|  - - - - - - - -
|  - - - - - - - -
|  *END

*END
```

Example: not allowed

```
|  *IF (Conditional)
|    - - - - - -
|    - - - - - -
|    - - - - - -
|
|  *IF (Conditional)
|    - - - - - -
|    - - - - - -
|    - - - - - -
|
|  *ELSE
|    - - - - - -
|    - - - - - -
|    - - - - - -
|
|  *ELSE
|    - - - - - -
|    - - - - - -
|    - - - - - -
|
|  *END
|    - - - - - -
|    - - - - - -
|    - - - - - -
|
|  *END
```

An *ELSE encountered in the skeleton is assumed to be matching the last *IF statement processed that has not already been matched with an *END, or, *ELSE and *END.

10.5.3.7.1. *IF Variable Conditional

Format:

$$*IF\ \Delta\ variable\text{-}name\ \Delta\ IS\ \Delta \left\{ \begin{matrix} SET \\ CLEAR \end{matrix} \right\} \Delta$$

Parameter:

variable-name	See 10.5.3.2 for description; may be supplied by a string, SGS reference, process parameter reference, or set reference that returns such a string.

Description:

Either a local or global variable may be tested for nonzero value (SET) or a zero value (CLEAR). When a search for the variable name is made, the local variable list is searched first from the most recently created backwards, then the global variable list is searched. The first variable found with the given name is tested.

Example:

Given the following skeleton and SGS:

```
    SKEL
1.  *DEFINE CHK
2.  *IF [#1] IS SET
3.  NONZERO VAUE FOR [#1]
4.  *ELSE
5.  ZERO VALUE FOR [#1]
6.  *END
7.  *END
8.  *CLEAR A
9.  *PROCESS CHK A
10. *INCREMENT A TO [VARSGS,1,1,2]
11. *IF [VARSGS,1,1,1] IS CLEAR
12. [VARSGS,1,1,1] = 0
13. *END
14. *LOOP
15. *IF A IS CLEAR
16. A IS ZERO
17. *END
    @EOF
    SGS
    VARSGS A,1
    @EOF
```

The generated output stream would look like:

```
ZERO VALUE FOR A
A IS ZERO
```

1. Statement 2 is false (the global variable A is 0), so Statement 3 between the *IF and *ELSE is skipped and Statement 5 between the *ELSE and *END is executed.

2. Statement 11 is false (the local variable A is 1), so Statement 12 between the *IF and *END is skipped.

3. Statement 15 is true (the global variable A is 0), so Statement 16 between the *IF and *END is executed.

10.5.3.7.2. *IF Existence Conditional

Format:

*IF Δ expression Δ

Parameter:

expression	May be a string (20 or fewer characters), an SGS reference, a process parameter reference, or a set reference.

Description:

The IF test checks to see if the given expression exists. If a string is supplied, the *IF statement is always true; since the string is there, it exists.

If the expression supplied is a process parameter reference, the test for existence is done to see if there was a process parameter of that number. Also, if that process parameter was supplied by an SGS reference, the test for existence is on the SGS reference.

If the expression supplied is an SGS reference, the following tests for existence are done:

*IF [l,n]	tests if there is an n^{th} statement with the label l.
*IF [l,n,f]	tests if there are at least f fields on an n^{th} statement with the label l.
*IF [l,n,f,S]	tests if there is at least an s^{th} subfield in an f^{th} field on an n^{th} statement with the label l.

Note that the IF test *IF [l] is not in the test for existence category. Since the reference [l] (which returns the number of SGSs with the label [l]) returns a numeric 0 rather than a 'no find' when there are no SGSs with that label, the *IF [l] is in the IF test for zero category (see 10.5.3.7.3).

If the expression supplied is a set reference, the existence test checks if the set referenced is defined and the n^{th} element entry in the set exists for the following expressions:

*IF [reserved-label,n]
*IF [reserved-label,n,f]
*IF [reserved-label,n,1,1]
*IF [reserved-label,n,2,1]

See 10.4.4 for description of set references.

Note that the IF test *IF [reserved-label] is not in the text for existence category since it returns a 0 rather than a 'no find' if there is no defined set with the name specified or no entries in that set. The *IF [reserved-label] is in the test for zero category (10.5.3.7.3).

Example:

Given the following SGSs and skeleton:

```
    SGS
    A  1,2,3  5,6
    @EOF
    SKEL
1.  *DEFINE TST
2.  *IF [#2]
3.  PROC PARA #2
4.  *ELSE
5.  NO PROC PARA #2
6.  *END
7.  *IF [#1]
8.  PROC PARA [#1]
9.  *END
10. *END
11. *IF [A,1,2,3]
12. SGS REF
13. *ELSE
14. NO SGS REF
15. *END
16. *PROCESS TST [A,1,3,1]
17. *PROCESS TST X [A,1,1]
    @EOF
```

The generated output stream would look like:

```
NO SGS REF
NO PROC PARA #2
PROC PARA #2
PROC PARA X
```

1. Statement 11 is false since there is no third subfield on field 2 of statement 1 with the label A. Therefore only statement 14 is executed.

2. Statement 16 processes TST.

3. Statement 2 is false since no process parameter 2 exists. Therefore, only statement 5 is executed.

4. Statement 7 is false since process parameter 1 was supplied by an SGS reference ([A,1,3,1]) that does not exist. Therefore, no statements between the *IF and *END are executed.

5. Statement 17 processes TST.

6. Statement 2 is true since process parameter 2 exists and the SGS reference supplied there exists. Therefore, only statement 3 is executed.

7. Statement 7 is true since process parameter 1 exists. Therefore, statement 8 is executed.

10.5.3.7.3. *IF Test for Zero Conditional

Format:

*IF Δ expression Δ

Parameter:

expression	May be VALUE OF numeric-expression, VALUE OF integer-expression, or a numeric expression (see 10.5.1).

Description:

If the numeric expression is zero. the test is false, otherwise the test is true.

Example:

Given the following SGS and skeleton:

```
    SGS
    NUM  4,6  9,12
    @EOF
    SKEL
1.  *DEFINE TST
2.  *IF +[#1]
3.  1 IS NONZERO
4.  *ELSE
5.  1 IS ZERO
6.  *END
7.  *END
8.  *IF [NUM]
9.  [NUM] ≠ 0
10. *END
11. *SET A TO [NUM,1,2,1]
12. *IF [*A]+4-12
13. NUMBER ≠ 0
14. *END
15. *PROCESS TST A
16. *PROCESS TST 0
    @EOF
```

The generated output stream would look like:

```
1 ≠ 0
NUMBER ≠ 0
1 IS NONZERO
1 IS ZERO
```

1. Statement 8 is true because the numeric SGS reference, [NUM], is ≠ 0. Therefore, Statement 9 is executed.

2. Statement 12 is true because [*A]+4-12 which is 1 is ≠ 0. Therefore, Statement 13 is executed.

3. Statement 15 processes TST and passes A as parameter 1.

4. Statement 2 is true because +[#1] which is the same as +A causes the variable A to be evaluated and the value is 9(9 ≠ 0). Therefore, Statement 3 is executed.

5. Statement 16 processes TST and passes 0 as parameter 1.

6. Statement 2 is false because +[#1] is the same as +0 (or a numeric 0). Therefore, Statement 5 is executed.

10.5.3.7.4. *IF Relational Tests

Format:

*IF Δ operand relation operand Δ

Parameters:

operand	May be 20 or fewer characters supplied by a string, process parameter reference, SGS reference, set reference, or numeric expression. If the string used contains a blank, period, semicolon, comma, left bracket, slash, plus, or minus, and evaluation of that string is not intended, pairs of apostrophes should be used around it ("..."). For a string where the apostrophes are to be included, single apostrophes should be used ('...').
relation	May be =, >, or <.

Description:

The first operand specified is compared to the second operand specified according to the relation: = tests for equality, > tests if the first operand is greater than the second operand; < tests if the first operand is less than the second operand.

There are two kinds of relational tests, symbolic and numeric. First both operands are evaluated (all numeric expressions and bracketed references satisfied). Then the condition of the first operand specified determines the kind of test done. If the first operand is a numeric expression, a numeric comparison is done. The second operand is always converted to be in the same condition as the first operand before the test is made. Since SSG operates internally in ASCII all symbolic comparisons are based on the ASCII collating sequence.

Example:

Given the following skeleton and SGS:

```
      SKEL
1.    *DEFINE TESTRS
2.    *IF [#1] > [#2]
3.    SYMBOLIC [#1] > [#2]
4.    *ELSE
5.    SYMBOLIC [#1] NOT > [#2]
6.    *END
7.    *END
8.    *DEFINE TESTRN
9.    *IF + [#1] > +[#2]
10.   NUMERIC [#1] > [#2]
11.   *ELSE
12.   NUMERIC [#1] NOT > [#2]
13.   *END
14.   *END
15.   *SET VAR TO [SGS2,1]
16.   *IF [SGSREL,1,1,1] > VAR
17.   SYMBOLIC [SGSREL,1,1,1] > VAR
18.   *ELSE
19.   SYMBOLIC [SGSREL,1,1,] NOT > VAR
20.   *END
21.   *IF +[SGSREL,1,1,1] > VAR
22.   NUMERIC [SGSREL,1,1,1] > VAR
23.   *ELSE
24.   NUMERIC [SGSREL,1,1,1] NOT > VAR
25.   *END
26.   *IF [*VAR] = 2
27.   *ELSE
28.   [*VAR] NOT 2
30.   *IF 2 < [SGS2,1]
31.   SYMBOLIC 2 < [SGS2,1]
32.   *END
33.   *SET F TO -4
34.   *SET E TO +1
35.   *PROCESS TESTRS F E
36.   *PROCESS TESTRN F E
      @EOF
      SGS
      SGSREL 666
      SGS2  A  B  C
      @EOF
```

The generated output stream would look like:

```
SYMBOLIC 666 NOT > VAR
NUMERIC 666 > VAR
3 NOT 2
SYMBOLIC 2 < 3
SYMBOLIC F > E
NUMERIC F NOT > E
```

1. Statement 16 is false because the symbolic string 666 left justified is not greater than (according to the ASCII collating sequence) the symbolic string VAR. Therefore, statement 19 is executed.

2. Statement 21 is true because the numeric value 666 is greater than the numeric value for VAR, which is 3 (see statement 15). Therefore, statement 22 is executed.

3. Statement 26 is false because the numeric value of VAR, which is 3, does not equal 2. Therefore, statement 28 is executed.

4. Statement 30 is true because the symbolic string 2 is less than (according to the ASCII collating sequence), the symbolic string 3. Therefore, statement 31 is executed.

5. Statement 35 processes TESTRS and passes two strings, F and E.

6. Statement 2 is true because the symbolic string F is greater than (according to the ASCII collating sequence) the symbolic string E. Therefore, statement 3 is executed.

7. Statement 36 processes TESTRN and passes two strings, F and E.

8. Statement 9 is false because the numeric value of F (which is –4) is not greater than the numeric value of E (which is +1). Therefore, statement 12 is executed.

10.5.3.7.5. *IF Row or Column Search Conditional

SGSs of a given label can be regarded as two dimensional tables. A row in the table corresponds to one particular statement. A column corresponds to one particular field and subfield in all the statements.

The following SGSs:

```
LABL    A,Q    Z,L,P   R6
LABL    B    S,T    Y
LABL    D
```

would look like the following, in a two dimensional table. See Table 10–2.

Table 10-2. Search Table Row/Column

Row	Column					
	field 1 subfield 1	field 1 subfield 2	field 2 subfield 1	field 2 subfield 2	field 2 subfield 3	field 3 subfield 1
stmt 1	A	Q	Z	L	P	R6
stmt 2	B		S	T		Y
stmt 3	D					

SGSs with the same label may be searched by row or column for a given image.

Format:

$$*IF\ \Delta \left\{ \begin{matrix} ROW \\ COLUMN \end{matrix} \right\} \Delta\ SEARCH\ \Delta\ FROM\ \Delta\ label \left[,start\text{-}stmt \left[,start\text{-}field \left[,start\text{-}subfield \right] \right] \right] \Delta\ FOR\ \Delta\ expression\ \Delta$$

Parameters:

label — See 10.3.1 for description of an SGS label; this label may be supplied by a string, SGS reference, process parameter reference, or set reference.

start-stmt — The statement number with the above given label where the search is to begin; may be supplied by a numeric expression or integer expression (see 10.5.1).

start-field — The field number on the above given statement with the given label where the search is to begin; may be supplied by a numeric expression or integer expression (see 10.5.1).

start-subfield — The subfield number in the above given field on the given statement with the given label where the search is to begin; may be supplied by a numeric expression or integer expression (see 10.5.1).

expression — The image that is to be searched for in the SGS tables; the images may be a string of 20 or fewer characters or the following references producing such a string: an SGS reference, process parameter reference, set reference, VALUE OF numeric expression reference (where the numeric value of the expression is taken), VALUE OF integer-expression, or a numeric expression. All numeric expressions are satisfied and the result is converted to a symbolic string for the search. If the string used contains a blank, period, semicolon, comma, left bracket, slash, plus, or minus, and evaluation of that string is not intended, pairs of apostrophes should be used around it ("..."). For a string where the apostrophes are to be included, single apostrophes should be used ('...').

Description:

The start-stmt, start-field, and start-subfield may be omitted, and if absent are assumed to have the value of one.

A row search starts from a particular statement, field, and subfield for a given label of SGS. The search goes from that point on the SGS to the end of the statement (row), comparing each subfield to the specified expression. If a match is found, the *IF statement is true and SSG sets the reserved global variables FLD and SFLD to the field and subfield values respectively, where the match is made. If no match is found, the *IF statement is false and the FLD and SFLD values remain unchanged.

A column search starts from a particular statement, field, and subfield for a given label of SGS. The search goes from that point on the SGS through the specified field and subfield of all the SGSs with the given label that follow. If a match is found, the *IF statement is true and SSG sets the reserved global variable CARD to the statement number where the match is made. If no match is found, the *IF statement is false and the CARD value remains unchanged.

The IF column (or row) search is more efficient than incrementing through lists of SGSs and doing an IF test for a match.

Example 1:

Given the following skeleton and SGSs:

```
      SKEL
1.    *SET VAR TO 1
2.    *SET GO
3.    *INCREMENT A TO [MIX] WHILE GO IS SET
4.    *IF COLUMN SEARCH FROM MIX,VAR FOR ASSIGN
5.    #ASG,T [MIX,CARD,2,1]
6.    *SET VAR TO CARD+1
7.    *ELSE
8.    *CLEAR GO
9.    *END
10.   *LOOP
      @EOF
      SGS
      MIX HOLD ALPHS
      MIX ASSIGN TEMP1
      MIX PROCESS FILE3
      MIX ASSIGN PF3
      MIX ASSIGN TEMP2
      @EOF
```

The generated output stream would look like:

```
@ASG,T TEMP1
@ASG,T PF3
@ASG,T TEMP2
```

The above IF column search, statement 4, searched through field one, subfield one of all the SGSs with the label MIX, starting with statement 1 the first time (initially VAR had the value 1). After a match was made, the value of CARD was set to the statement number where the match was found and the *IF was true, so statement 5 and 6 were executed. An assign card was generated and VAR was set to the next statement after the match statement (for the next loop through the IF column search). When no more assign images were found, the IF column was false and statement 8 was executed and the global variable GO was cleared. Thus, the WHILE phrase of the increment loop became false and incrementing was stopped. The increment, statement 3, merely insured that the IF column search was done at least as many times as there were MIX cards for the case where every MIX card had an ASSIGN in field one, subfield one. The WHILE phrase on the increment allowed early termination of the loop for the case where the entire list was searched before the variable A had the value [MIX].

Example 2:

Given the following SGSs and skeleton:

```
    SGS
    ALL AVAIL FILES  T3,P4,XR2,PF$,TEMP4
    FILES ARE FOUND ON ALL
    @EOF
    SKEL
1.  *IF ROW SEARCH FROM [FILES,1,4,1],1,3,1 FOR PF$
2.  PF$ WAS FOUND ON THE SGS [FILES,1,4,1]
3.  STATEMENT 1,FIELD [*FLD] AND SUBFIELD [*SFLD]
4.  *END
    @EOF
```

The generated output stream would look like:

```
PF$ WAS FOUND ON THE SGS ALL
STATEMENT 1,FIELD 3 and SUBFIELD 4
```

Statement 1 does an IF row search on the first SGS with the label ALL starting at field 3, subfield 1 and searches to the end of that SGS until a match is found with the image PF$. The *IF is true and statements 2 and 3 are executed. Also the variables FLD and SFLD are set to 3 and 4, respectively.

10.5.3.7.6. *IF CORRECTION ENTRY EXISTENCE

Format:

*IF Δ element-name [/version-name] Δ HAS Δ [{ PERM / TEMP } Δ] CORRECTIONS Δ

Parameters:

element-name }	See 10.5.3.3.1 for description
version-name }	Refers to an element-entry name.

Description:

The search may be made on the PERM (PCF set) or TEMP (primary TCF set) or both sets (by omitting PERM or TEMP) to see if the specified element entry exists. If PERM or TEMP is omitted, a search of both sets occurs and the existence of the specified element entry in either set is sufficient for the statement to be true. PCF or primary TCF entries found cause the *IF to have a true value. Just a *card in an input stream (or the existence of a program file element when using P or T options) causes an element entry to be created. However, if an element entry is created via the *CREATE directive, the *IF test for that entry's existence would be false since SSG flags that entry as a special case having no input associated with it.

Example:

Given the following primary temporary corrections, permanent corrections, SGS, and skeleton:

```
TEMP COR
*ELEA
-212
*ELEB/VER1
*ELEC
-6,12
@EOF
PERM COR
*ELEB/VER1
-1,3
*ELED
-45,61
@EOF
SGS
E ELEC
@EOF
SKEL
1.  *IF ELEA HAS CORRECTIONS
2.  ELEA PRESENT
3.  *ELSE
4.  ELEA NOT PRESENT
5.  *END
6.  *IF [P,1,1,1]/[P,1,2,1] HAS TEMP CORRECTIONS
7.  [P,1,1,1]/[P,1,2,1] HAS TEMP
8.  *END
9.  *CREATE PERM:  ELEC
10. *IF [E,1,1,1] HAS PERM CORRECTIONS
11. [E,1,1,1] HAS PERM
12. *END
    @EOF
```

The generated output stream would look like:

```
ELEA PRESENT
ELEB/VER1 HAS TEMP
```

1. Statement 1 searched both the PCF and primary TCF set for the element entry ELEA and found it in the primary TCF. Therefore, the *IF was true and statement 2 was executed.

2. Statement 6 searched the primary TCF set for the element entry ELEB/VER1 and found it (note: only *ELEB/VER1 was necessary in the run stream to cause an element entry to be created). Therefore, the *IF was true and statement 7 was executed.

3. Statement 10 searched the PCF set for the element entry ELEC and found that that entry had been created by the *CREATE directive. Therefore, the *IF was false and statement 11 was skipped.

10.5.3.7.7. Compound *IF Statements Using Boolean Operators

The conditional tests on *IF statements may be compounded to create Boolean expressions. A Boolean expression is made up of *Boolean-operand Boolean-operator Boolean-operand,* and gives a true or false result. The Boolean operators are AND, OR, XOR. Boolean operands are any of the conditionals described under 10.5.3.7 or Boolean expressions made up of those conditionals. All boolean expressions are evaluated left to right taking two operands at a time and applying the result to the next operand found.

The Boolean operator, NOT, may also be used in *IF statements and applies only to the conditional immediately following it. A logical NOT is done on the results of the conditional.

The results of the logical operations done using AND, OR, XOR are shown in Table 10-3.

Table 10-3. Logical Operations Using AND, OR, XOR

1st Boolean Operand	2nd Boolean Operand	Result of AND	Result of OR	Result of XOR
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

Example:

The IF statement:

```
*IF B IS SET AND VALUE OF A = -1 OR NOT [X,1,1,1] = STOP
```

does the following operations, in order:

1. test if variable B > 0
2. test if variable A = -1
3. result of (1) AND result of (2)
4. test if [X,1,1,1] = STOP
5. NOT the result of (4)
6. result of (3) OR result of (5)

The result of (6) is the value of the *IF test.

The IF statement:

```
*IF NOT ELEA HAS CORRECTIONS AND COLUMN SEARCH FROM;
LAB,6,3,4 FOR ELEA
```

does the following operations, in order:

1. test if the PCF set or primary TCF set has the element entry ELEA
2. NOT the result of (1)
3. do a column search beginning with the sixth SGS with the label LAB, field 3, subfield 4 for the image ELEA
4. result of (2) AND result of (3)

The result of (4) is the value of the *IF test

NOTE:

*The *IF statement may be continued using the standard continuation character, a semicolon.*

10.5.3.8. Merging Permanent and Temporary Streams

See 10.4 for description of permanent and temporary streams. The Permanent Correction File (PCF) set is designed as a means of maintaining groups of correction images (element entries) to some group of base symbolic elements. See 1.2 on modifying symbolic elements and the description of format for line correction statements. The Temporary Correction File (TCF) sets are designed as a means of changing or adding corrections to an existing PCF set element entry. A TCF set entry may contain two types of line corrections, those relative to the base symbolic elements being modified (additions to the PCF set element entry) or those relative to an already existing line correction in the PCF set element entry. Both correction types can be intermixed in a TCF set element entry.

Those TCF set corrections that are relative to the base symbolic elements follow the format described in 1.2. Those TCF set corrections relative to an already existing PCF set line corrections may have the following formats:

-perm-line-nos/relative-line-nos

or

-perm-line-nos/relative-line-nos/new-line-nos.

where the perm-line-nos matches (is the same as) the line correction in the PCF set element entry being altered; the relative-line-nos are the line numbers where a change is to be made to the PCF line correction.

For relative referencing purposes, the PCF line correction (as specified) is relative line zero and all images following that line correction until the next line correction in that element entry are relative lines 1,2,...etc. Relative line numbers may be of two forms:

-n
-n,m

where –n indicates that the data following that relative correction is to be inserted after relative line number n in the PCF element entry. –n,m indicates that relative lines n through m (where m > n) inclusive, are to be removed and any data following that relative correction is to be inserted in the PCF element entry at that point. Relative line corrections to the PCF may not be partial line corrections. New–line–nos may be of three forms:

```
p
p,q
p,q-
```

These line corrections cause –p, –pq, or –pq–, respectively, to be inserted at the point that the relative line correction is being made in the PCF element entry before any associated data following the relative line correction is inserted.

Example 1:

The following shows how line corrections in a PCF element entry are numbered for relative referencing:

Corrections	Relative line numbers
-2,3	0
A	1
B	2
C	3
-10	0
XYZ	1
-14,16	0

Therefore, any line in a PCF element entry may be referenced first according to the line correction number associated with it, and then according to its relative position to that correction line.

Example 2:

Given the following PCF set and primary TCF set input:

```
PERM COR
*A
-4,6
X
YYY
-10
B
-25,25-
/ABC/DEF/
-32,33-
/L/S/
/AO/A2/
-43,44
-72,76
@EOF
TEMP COR
*A
-4,6/2,2
LJK
```

```
-10/0,0/12,12
NEW
-25,25-/1,1
/ABC/EFJ/
-32,33-/0,0/32,32-
-32,33-/1/35,35-
-54
QRS
@EOF
```

The corrections in the primary TCF set would reference, as follows:

1. -4,6/2,2
   LJK

   Deletes line 2 relative to the correction line -4,6 in the PCF, which is YYY, and inserts the line LJK at that position.

2. -10/0,0/12,12
   NEW

   Deletes line 0 relative to the correction line -10 in the PCF, which is -10, and inserts the image -12,12 followed by the line NEW.

3. -25,25-/1,1
   /ABC/EFJ/

   Deletes line 1 relative to the correction line -25,25- in the PCF, which is /ABC/DEF/, and inserts the line /ABC/EFJ/ at that position.

4. -32,33-/0,0/32,32-

   Deletes line 0 relative to the correction line -32,33- in the PCF, which is -32,33-, and inserts the image -32,32-.

5. -32,33-/1/35,35-

   Inserts the image -35,35- following line 1 relative to the correction line -32,33- in the PCF, which is /L/S/.

6. -54
   QRS

   Since this is not a relative line correction, it must be a correction to be added to the PCF. Therefore, -54 and QRS are inserted between -43,44 and -72,76 in the PCF.

10.5.3.8.1. Merging PCF and Primary TCF Element Entries (*CORRECT)

Format:

```
*CORRECT [,options] Δ element-name [/version-name] Δ [ {PERM} Δ ]
                                                     {TEMP}
[ ---------- ]
[ ---------- ] corrections-from-skel
[ ---------- ]

*END
```

Parameters:

options

- none — The images resulting from the merge are sent to the generated output stream.
- P — The images resulting from the merge are sent to a program file's symbolic element with the same name as the element entry. This option may only be used when the P option is used on the @SSG call and the PCF had its input defined by symbolic elements in a program file (see 10.4.2, second paragraph). The program file updated when doing a *CORRECT,P is the same one that is specified for the PCF set input. This is the only case where SSG alters an input file (PFI$ is used to perform the insert).
- K — Used in conjunction with the P option to cause the images resulting from the merge to be also sent to the generated output stream.

element-name, version-name — See 10.5.3.3.1 for description: specifies the name of the element entry from which the corrections are to be taken. May be supplied by a string, set reference, SGS reference, or process parameter reference.

Description:

If PERM is specified, SSG takes only the element entry corrections from the PCF set. If TEMP is specified, SSG takes only the element entry corections from the primary TCF set. If neither PERM or TEMP are specified, SSG takes both the PCF set and primary TCF element entries or whichever exist. The specified element entry corrections desired are merged along with corrections from the skeleton (if any). Corrections from the skeleton are those images between the *CORRECT and the matching *END. Regardless of the other source of corrections, any corrections from the skeleton are always used in the merge. If any element entry or corrections that are specified or assumed, are not present for the *CORRECT, the merge consists of the remaining specified or assumed entries.

When using the P option on the @SSG call (a program file is specified on the @SSG call as the source for the PCF set), a *CORRECT,P is ignored for an element entry that does not exist in the PCF set (there is no symbolic element with that name in the specified program file). An *CREATE PERM must first be done for that element entry to indicate to SSG that a new symbolic element is to be created in the specified program file when doing the *CORRECT,P. The normal case is where an element entry in the PCF set (existing symbolic element in the given program file) is corrected and re-inserted into the given program file, simulating an update process. The P option is also ignored if TEMP is specified on the *CORRECT,P statement too.

When a *CORRECT is done with PERM specified, revised temporary corrections are not created since no primary TCF corrections are used.

Conflicts in line numbers are flagged (see 10.6 for diagnostic messages). TCF set corrections override PCF set corrections which override corrections from the skeleton, if a conflict occurs. Since conflicts that cause overrides are not fatal errors, such a feature may be used to cause desirable changes in the resulting output. When a conflict does occur the corrections from the set with the highest priority are saved as part of the resulting output and the corrections with the lowest priority are printed in the override message and are ignored.

Example:

Given the following runstream:

```
     @ASG,T  PF
     @ELT,I  PF.ELEA
     -2,2
     XYZ
     -4,7
     @EOF
     @ELT,I  PF.ELEC
     -10
     RST
     -12
     PQR
     @EOF
     @SSG,KBP ,,,,,,PCF/1,PF.
     TEMP COR
     *ELEA
     -2,2/1
     LMN
     *ELEB/ONE
     -6,10
     *ELEC
     -10/1,1
     JKL
     RVW
     @EOF
     SKEL
1.   ONE XXX
2.   *CORRECT ELEA PERM
3.   -10,10
4.   *END
5.   *CREATE PERM:  ELEB/ONE
6.   *CORRECT,P ELEB/ONE
7.   -42,42-
8.   /AA/BB/
9.   *END
10.  TWO XXX
11.  *CORRECT,PK ELEC
12.  *END
     @EOF
     @EOF
```

The generated output stream would look like:

```
ONE XXX
-2,2
LMN
XYZ
-4,7
-10,10
TWO XXX
-10
JKL
RVW
-12
PQR
```

At the end of the SSG execution the program file, PF., has three symbolic elements, ELEA.,ELEB/ONE, and ELEC., and their contents are:

ELEA	ELEB/ONE	ELEC
-2,2	--6,10	-10
XYZ	--42,42-	JKL
-4,7	/AA/BB/	RVW
		-12
		PQR

1. Statement 2 causes the PCF set element entry, ELEA, to be merged with the corrections from the skeleton (those between the *CORRECT and *END), and the resulting images were sent to the generated output stream. PF.ELEA is not updated (changed) because a *CORRECT,P was not done.

2. Statement 6 caused the primary TCF set element entry, ELEB/ONE, to be merged with the PCF set element entry ELEB/ONE, (since it is a created entry, same as merging with nothing) and the corrections from the skeleton. Since the P option is on the *CORRECT, the resulting images from the merge are inserted as a symbolic element, ELEB/ONE in the PCF set program file, PF. Note that since there was no element entry in the PCF set existing (on input) with the name ELEB/ONE, a *CREATE PERM (statement 5) was necessary before SSG would recognize the *CORRECT,P. Since the P option was present and the K option was not (on the *CORRECT) the resulting images were not sent to the generated output stream.

3. Statement 11 caused the PCF set element entry, ELEC, to be merge with the primary TCF element entry, ELEC. There are no corrections from the skeleton (nothing between the *CORRECT and *END). Since the P and K options are on the *CORRECT, the resulting images from the merge are inserted as a symbolic element, ELEC in the PCF set program file, PF, and are sent to the generated output stream. When the new symbolic element, ELEC, is inserted into PF the old ELEC is deleted.

10.5.3.8.2. Merging TCF Element Entries (*MERGE)

Format:

*MERGE [,options] Δ element-name [/version-name] Δ tcf-set-name Δ WITH Δ

tcf-set-name Δ [GIVING Δ tcf-set-name Δ]

Parameters:

options	R	See below for description.
	K	Used when the GIVING phrase is on the *MERGE, to cause the resulting images to be sent to the generated output stream.
element-name version-name		See 10.5.3.3.1 for description of element entry names. May be supplied by a string, set reference, SGS reference, or process parameter reference.
tcf-set-name		Specifies the TCF sets where the element entries are to be taken from for the merge, and, in the GIVING phrase, specifies the TCF set destination of the resulting element entry, (may be the primary or any secondary TCF set). See 10.4.3 for description of the TCF set names. May be supplied by a string, set reference, SGS reference or process parameter reference. (The TCF set name for the primary TCF set is TCF.)

Description:

*MERGE merges the specified element entry from the first TCF set given with the elment entry of the same name from the second TCF set given. The GIVING phrase is optional. If it is omitted, the resulting images from the merge are automatically sent to the generated output stream. If the GIVING phrase is present, the resulting images from the merge are inserted as an element entry (of the same name specified) into the TCF set given in the GIVING phrase. When the GIVING phrase is present the K option must be used on the *MERGE if the merge images are also to be sent to the generated output stream.

When the GIVING phrase is present on the *MERGE directive, SSG creates an internal element entry, associates the resulting merge images with it, and attaches it to the TCF set specified in that phrase. If an element entry of the name already exists in that TCF set, the old entry is lost and all references apply to the newly created element entry. The original TCF input files are never altered, and any internal element entries that SSG creatos are lost when SSG terminates.

R OPTION

With no R option on the *MERGE directive, SSG combines the two TCF sets' element entries. Both entries' line corrections are based on the same PCF and base symbolic element. The corrections, both to PCF (relative corrections) and in addition to a PCF (base symbolics corrections) are put in sequential order. (Each entry correction must be in sequential order before the merge.) All sequence errors are flagged and the line corrections discarded. The first TCF set entry specified on the *MERGE takes precedence over the second TCF set entry in the case of line number conflicts. One TCF set entry may not do relative correction to another. All relative corrections are assumed to be based on a PCF set entry.

With the R option on the *MERGE directive, a special type of merge is done between the two TCF set element entries specified. SSG assumes that the second TCF set element entry (from second TCF set specified) was merged with a PCF set element entry and a revised PCF set element entry was created. SSG also assumes that the line numbers in the first TCF set element entry (from first TCF set specified) are based on the created revised PCF set element entry. SSG does the merge, changing the line numbers from the first TCF set element entry and merging them with the second TCF set element entry so that all line numbers in the resultant merge are relative to the original PCF set element entry.

With the *MERGE directive, element entries with the same name from any number of TCF sets may be merged, two at a time. By allowing the results of the final merge to end up in the primary TCF set, a *CORRECT may be done to merge PCF corrections with the TCF set corrections.

Example 1:

Given the following runstream:

```
      PERM COR
      *A/VA
      -2,5
      A
      B
      C
      -10
      D
      E
      @EOF
      TEMP COR
      *A/VA
      -2,5/1,1
      A1
      L1
      -2,5/3,3
      C1
      -10/0,0/12
      @EOF
      TEMP COR : TCFNEW
      *A/VA
      -1,1
      R
      -10/2,2
      E1
      F
      @EOF
      SKEL
1.    STEP 1
2.    *MERGE,K A/VA TCFNEW WITH TCF GIVING TCF
3.    STEP 2
4.    *CORRECT A/VA
5.    *END
      @EOF
```

Both A/VA in the primary TCF set input and A/VA in the second TCF set input, TCFNEW, have their line corrections based on A/VA in the PCF set and some base symbolic element (which the PCF set entry, A/VA, is also based on).

The generated output stream would look like:

```
STEP 1
-1,1
R
-2,5/1,1
A1
L1
-2,5/3,3
C1
-10/0,0/12
-10/2,2
E1
F
STEP 2
-1,1
R
-2,5
A1
L1
B
C1
-12
D
E1
F
```

1. After statement 2, and while SSG is still processing, the primary TCF set entry A/VA contains the same images as those seen between STEP 1 and STEP 2, exclusively, in the generated output stream. The K option was necessary on the *MERGE to cause the images to go to the generated output stream, since the GIVING phrase was present.

2. When statement 4 was executed, element entry A/VA was taken from both the PCF set and the primary TCF set (getting the internally created entry) for the merge.

3. Thus the final result was to merge all existing corrections for A/VA to later apply to some base symbolic element.

Example 2:

Given the following input runstream:

```
SKEL
*CORRECT B/XVERSION
*END
@EOF
PERM COR
*B/XVERSION
-6,8
XXXX
-12
R
S
@EOF
TEMP COR
```

```
*B/XVERSION
-6,8/1,1
YYY
ZZZ
-12/0,0/13
@EOF
```

The generated output stream would look like:

```
-6,8
YYY
ZZZ
-13
R
S
```

The above corrections in the generated output stream could be called revised PCF corrections since, they are the result of TCF corrections applied to PCF corrections. Assume that the above listing of revised PCF corrections was the only available list and additional changes were necessary. A TCF set entry could be created that had its line numbers based on the above revised PCF listing, as follows:

```
-6,8/1,2
NEWXXX
-13/1,1
-20,21
```

Then a new SSG call could be made using the same PERM COR and TEMP COR input as before plus the new line numbers in a TCF set, TCFCOR, and a slightly revised skeleton, as follows:

```
    SKEL
1.  *MERGE,R B/XVERSION TCFCOR WITH TCF GIVING TCF
2.  *CORRECT B/XVERSION
3.  *END
    @EOF
    PERM COR
    *B/XVERSION
    -6,8
    XXXX
    -12
    R
    S
    @EOF
    TEMP COR
    *B/XVERSION
    -6,8/1,1
    YYY
    ZZZ
    -12/0,0/13
    @EOF
    TEMP COR : TCFCOR
    *B/XVERSION
    -6,8/1,2
    NEWXXX
    -13/1,1
    -20,21
    @EOF
```

The generated output stream would look like:

```
-6,8
NEWXXX
-13
S
-20,21
```

1. Statement 1 causes the element entry B/XVERSION in TCFCOR to be merged with the entry of the same name in TCF (the primary TCF set) knowing that TCFCOR's line corrections are based on revised PCF corrections that TCF's corrections helped to create. Thus, the line corrections in the TCFCOR entry are modified to be relative to the orginal PCF entry and are merged with the TCF entry corrections. The new element entry created and attached to the primary TCF set looked like the following:

```
-6,8/1,1
NEWXXX
-12/0,0/13
-12/1,1
-20,21
```

2. Statement 2 caused the new element entry attached to the primary TCF set to be merged with the PCF set entry of the same name. Thus, the above corrections in 1. were applied to the correction that came in with PERM COR and the desired result is seen in the generated output stream.

10.5.3.8.3. Change Control Characters

The standard control character for the *CORRECT and *MERGE merges is the minus (-). This standard may be changed to another control image (up to three characters long), according to the following format:

```
- = new-control-characters
```

No control characters may be numeric and an indirect reference is not allowed.

When doing a *CORRECT type merge, the change of control character image may appear as the first image of a PCF entry, as the first image of a primary TCF entry or immediately after the *CORRECT in the skeleton. Any other such images elsewhere in the entries or between the *CORRECT and *END, that look like change control character images are treated simply as nondirective images and do not affect the control character.

When the change control character image is the first one in the PCF entry it triggers the change, and the image is passed to the revised PCF entry (if one) but does not appear in the output stream. If the change control character image is from the primary TCF entry, it triggers the change, and the image is passed to the revised TCF entry, but is not passed to the revised PCF entry (if one) nor the output stream. If the change control character image appears after the *CORRECT in the skeleton, it triggers the change but does not appear in the revised PCF, revised TCF or output stream.

When a change control character image for a *CORRECT comes from more than one source the destination for each image is still as described above. However, the new control characters for that merge are determined according to the following priority: TCF overrides PCF and skeleton; PCF overrides skeleton. After each *CORRECT is done (the *END is encountered) the control character is reset to -.

When doing a *MERGE type of merge, the change control character image may appear as the first image of a TCF set entry. If an image appears in both entries, the one from the first specified set is used. This image triggers the change of control chracters and is sent to the third specified TCF set (if one) but does not appear in the output stream. When the merge is complete, the control character is reset to –.

10.6. DIAGNOSTIC MESSAGES

The following is a list of only those processor messages that are not self-explanatory.

******** tcfset STREAM TOTAL IMAGES : n ********

This message is printed immediately following each TCFset-G option printing (is printed only if the G option is present on the SSG call). tcfset is the setname and n is the total number of correction images for that set. (*element/version cards not included).

SPECn message

Where 'n' is the parameter field number on the processor call statement (0 if from runstream) and 'message' may be any of the following:

NOT PCF,TCF,SGS	SSG expected PCF, TCF (or TCF set name), or SGS and did not find it.
/VER NOT NUMERIC	Following PCF, TCF, tcf-set-name or SGS a / <numeric>is expected and not found.
NOT PROGRAM FILE	SSG expected the specified file to be a program file and it was not.
FILE EMPTY	SSG expected input from a file and it was empty.
BAD LABEL IMAGE	The label image of the specified file is bad.
PF ELE NOT FOUND	The specified element was not found in the given program file.
PF CYCLE ERROR	The cycle given for the specified program file is in error.
PF ELE DELETED	The specified element has been deleted from the given program file.
USE OR ASG ERROR	An error occurred trying to do an @USE or @ASG on the specified file.

INTERNAL SSG FILE COULD NOT BE ASSIGNED

An internal file SSG uses for processing could not be assigned.

DUPLICATE n IN { PCF / TCF / tcf-set-name }

When ordering the input element/version entry, 'n', a duplicate name was found in the specified set (PCF, TCF or TCF set). The second entry is discarded.

POSSIBLE ERROR ON FOLLOWING * CARD-ASSUMED TO BE DATA

When SSG is reading input, temporary and permanent element/version entries and an * card is encountered which has characters other than alphabetics, numerics, – or $ this message is printed. The * card image is then assumed to be part of the last * card data.

ADDR NOT FOUND FOR WRITE TO REV SKEL FILE

This message is printed when a program file element is specified as the output elment for the revised skeleton (param–5) and an error is returned from ER PFWL$.

ERROR IN REVISED SKELETON – SGS CANNOT PROCESS IT

This message is printed at the end of the revised skeleton listing when the E option is used (no D option) and an error in the skeleton is detected. Another message explaining the error is printed in the revised skeleton listing where the error occurs.

TRUNCATED IMAGE EXCEEDED 80 CHAR MAX

This message is printed when an image that is greater than 80 characters is read from an input file. The image is truncated at 80 characters and SSG continues to process it.

FILE IDENTIFICATION STATEMENT ERROR

When the file identification statement, which specified input streams, has a format error or specifies an illegal input type, the above message is printed.

SKELETON IS ABSENT

The skeleton is the only mandatory input stream since it directs the processor. Its absence is noted by the above message.

ERROR IN FORMATION OF OUTPUT RUNSTREAM – SSG WILL NOT PERFORM DYNAMIC @ADD

The absence of the B option signals the generation and execution of an output runstream. If an error is detected during a non–B option generation, the above message is printed and the generated runstream will not be dynamically @ADDed. The programmer may perform his own execution mechanism for the generated runstream. Usually, sequence errors found during merging operations causes this condition.

ERROR IN BREAKPOINT n AT SKEL LINE x

If a format error exists on the BRKPT directive, the above message results where 'n' is the part number of the last generated output stream and x is the SKEL line number of the *BRKPT.

NO–FIND ON FOLLOWING IMAGE

The above message indicates a reference by the following image which couldn't be satisfied.

HAVE ENCOUNTERED 100 NO–FIND RETURNS – NO–FIND MESSAGES TO BE SUPPRESSED

Upon the occurrence of 100 unsatisfied references, the above message is printed and no further unsatisfied reference messages will be printed.

FORMAT ERROR IN FOLLOWING CARD

THE FOLLOWING CARD IS OUT OF SEQUENCE

SEQUENCE OR FORMAT ERROR ON FOLLOWING CARD

Above messages are printed when the specified conditions occur.
Sequence messages refer to a merge being done at the time (*CORRECT or *MERGE).

CONFLICT IN REL CORR:n MERGE WITH p q

During the merging of streams with *MERGE, internal conflicts in the correction numbers may occur. Some conflicts are of such a nature that they cannot be resolved (due to ignorance of the PCF entry involved). As a result SSG prints one of the above messages and discards the correction from the first TCF set, 'n'. 'q' is the element/version entry and 'p' is the second TCF set.

I/O ERROR p AT LOCATION n

The above message signals an I/O error of the type 'p'. (n is in decimal.)

INCORRECT ATTEMPT TO MAKE REL.COR.n

During the merging and modifying of the input streams, with *CORRECT, the occurrence of a modifying image (relative correction to permanent stream) in the temporary stream which is badly formatted or has bad line correction numbers results in the above message where 'n' is the element/version entry being merged.

FOLLOWING REL TEMP COR TRUNCATED

If during the merging and modifying of TCF streams with *MERGE, a relative correction or image that must be modified exceeds 80 characters, it is truncated and the above message printed.

TEMPORARY CORRECTIONS OVERRIDE { PERMANENT / SKELETON GENERATED } CORRECTIONS IN ELEMENT n

PERMANENT CORRECTIONS OVERRIDE SKELETON GENERATED CORRECTIONS IN ELEMENT n

During the merging of the input streams, conflicts in the line correction numbers may result. The conflicts are settled by the priority, temporary overrides permanent which overrides skeleton. Upon a conflict, the appropriate message is printed where 'n' is the element/version entry being merged. The overridden correction images are also printed out. The programmer should take note of any conflicts between permanent and skeleton streams.

ADDR NOT FOUND FOR WRITE TO PERM COR PF

SSG prints the above message if the PCF input is from a program file (P option) and if a *CORRECT,P is done, and an error is returned trying to obtain the PCF program file's next write address.

BAD IMAGE CONTROL WORD n:INTERNAL FILENAME p

The above message is printed if, while reading input streams, SSG receives an image control word, 'n', that appears incorrect. The internal filename where the error is detected is printed, 'p'. See Volume 4-2.6.4.2 for explanation of image control words.

11. File Structure and Maintenance

11.1. INTRODUCTION

This section describes the file table formats and file table maintenance software, both of which are normally transparent to the user. The information is provided:

- To give insight into the file structure used by the FURPUR processor, the language and system processors, and the symbiont complex;
- To enable the user to write additional software to build, insert, and retrieve data from the files.

The operating system generates three major types of files:

1. Program File
2. Element File
3. System Data Format (SDF) File

The format of each of these files and the manner in which they are manipulated are described in the following paragraphs:

11.2. FILE FORMATS

11.2.1. Program File Format

A program file can be defined as a partitioned random access file consisting of a group of elements residing on mass storage. A program file may contain symbolic, relocatable, omnibus, or absolute elements or a combination of elements. It may be either a temporary or a catalogued file. Since the elements are named, they may be manipulated on an individual basis. Thus, the elements needed to produce an executable program may be collected from one program file or from several program files.

It must be emphasized that while the program file is logically structured as shown in Figure 11-1, physically the elements that make up the file are not necessarily contiguous. Linkages are automatically generated by the Executive to logically structure the file as a separate continuous entity.

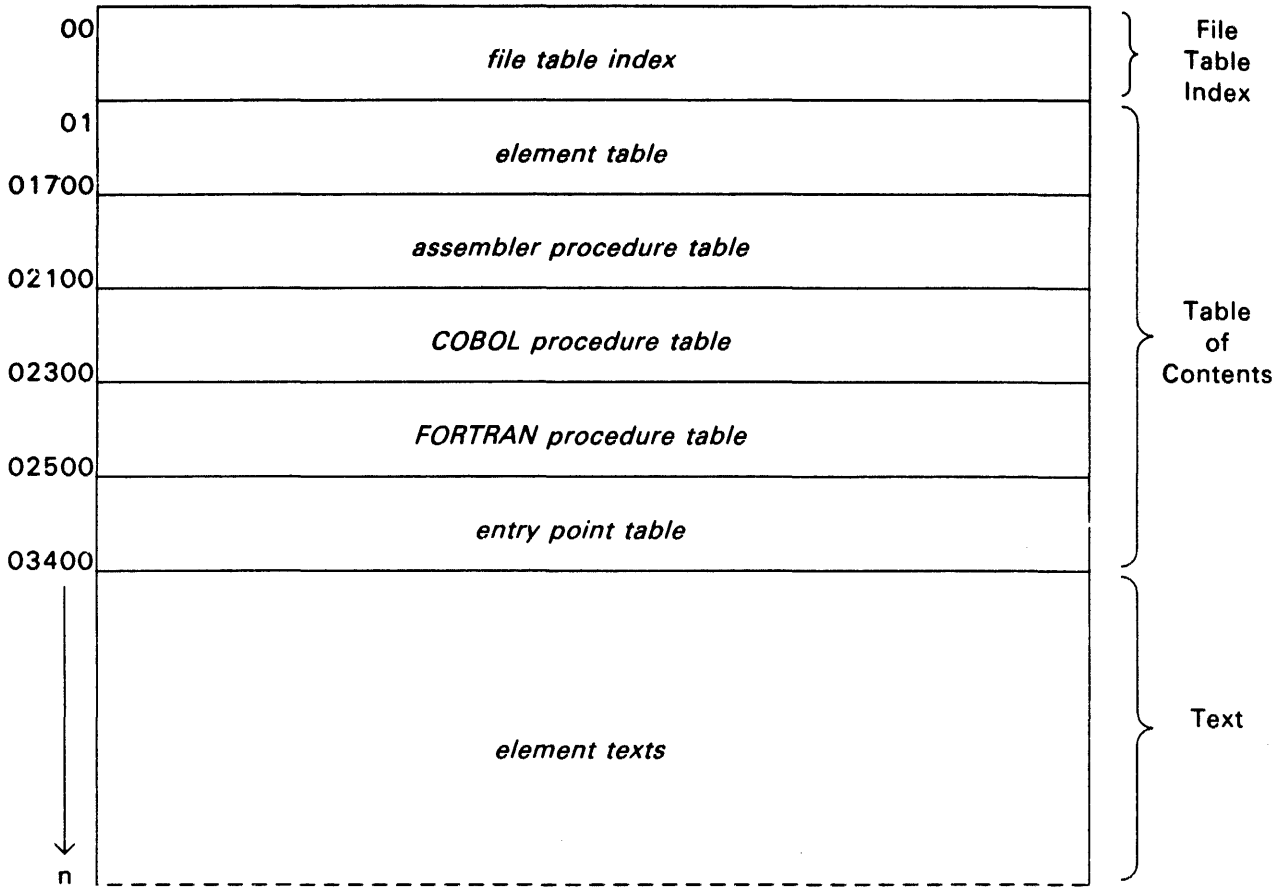


Figure 11-1. Program File Format

The program file (see Figure 11-1) has three major sections:

1. File Table Index – Contains pointers and links (relative to the beginning of the file) to the tables which comprise the file's table of contents (see Figure 11-2).

2. Table of Contents – Provides pointers to the elements (element table), procedures (procedures tables), and entry points for relocatable binary elements (entry point table). These tables are illustrated and described in the following paragraphs. Each table, except for the element table which always starts at sector 1, begins at its system standard sector, or at a greater sector if a table has extended over the standard starting sector, or at a lesser sector if the file has been @PACKed with the M option.

3. Text – The elements.

Word	35	24	18 17	0
0	* *	P	F	* *
1	*next sector available for writing text*			
2	*run-id (used by CTS)*			
3	*sector address of segment in main storage*	*change indicator*	*starting sector address of table on mass storage*	
4	*length of table in words*		*starting address of buffer*	
5	*pointer table length -1*	*item size*	*end address + 1 of buffer*	
6-8	*similar information for assembler procedure table*			
9-11	*similar information for COBOL procedure table*			
12-14	*similar information for FORTRAN procedure table*			
15-17	*similar information for entry point table*			
18	*0*		*new relocatable element*	
19	*sequence number of latest absolute element in file*			
20	*0*			
26	*0*			
27	*TIME$ value (used by CTS)*			

See 1:2-74

NOTE:

The File Table Index is initially zero filled.

Figure 11-2. File Table Index Format

Word 0	This word, when as indicated, is used to identify the file as a program file.
Word 1	Next available sector at which text may be written.
Word 2	Run-id (used by CTS, see Volume 2-1.4.10).
Word 3-5	Element Table information. Change Indicator – Set nonzero to indicate that the table segment presently in the buffer has been modified and should be written back to mass storage.
Word 6-8	Assembler procedure table information.
Word 9-11	COBOL procedure table information.
Word 12-14	FORTRAN procedure table information.
Word 15-17	Entry point procedure table information.
Word 18 (S6)	1 – Relocatable element added to program file 0 – Absolute element added to program file Allows automatic remapping of TPF$ for @XQT when new relocatable elements have been added.
Word 19	Sequence number of last absolute element added to the program file. This will be zero if there are no absolute elements in the program file or if the last absolute element added has been deleted.
Word 27	TIME$ (used by CTS, see Volume 2-4.5.3).

11.2.1.1. Element Table

The element table (see Figure 11-3) contains the Fieldata element and version name of each element, its type (symbolic, relocatable, omnibus, or absolute), and pointers to its text within the program file. It also provides information concerning

- the size of the element,
- the time and date the element was created,
- the sequence number of the element in the file which is used for linking entries within the element table (the sequence number specifies the order in which the element texts are entered in the file), and
- the address of the text.

The element table has the first 140 words (5 sectors) reserved as a pointer table and control word. The first 139 words are used to hold pointers that start chains. The elements that belong to a particular chain are determined by dividing the element's name by the length of the pointer table (139) and using the remainder as an index into the pointer table. Actually 278 chains may exist as each word contains two pointers; the pointer in H1 is used if the quotient was odd, and the pointer in H2 is used if it was even. The control word contains the item size in H1 and the total number of entries in H2.

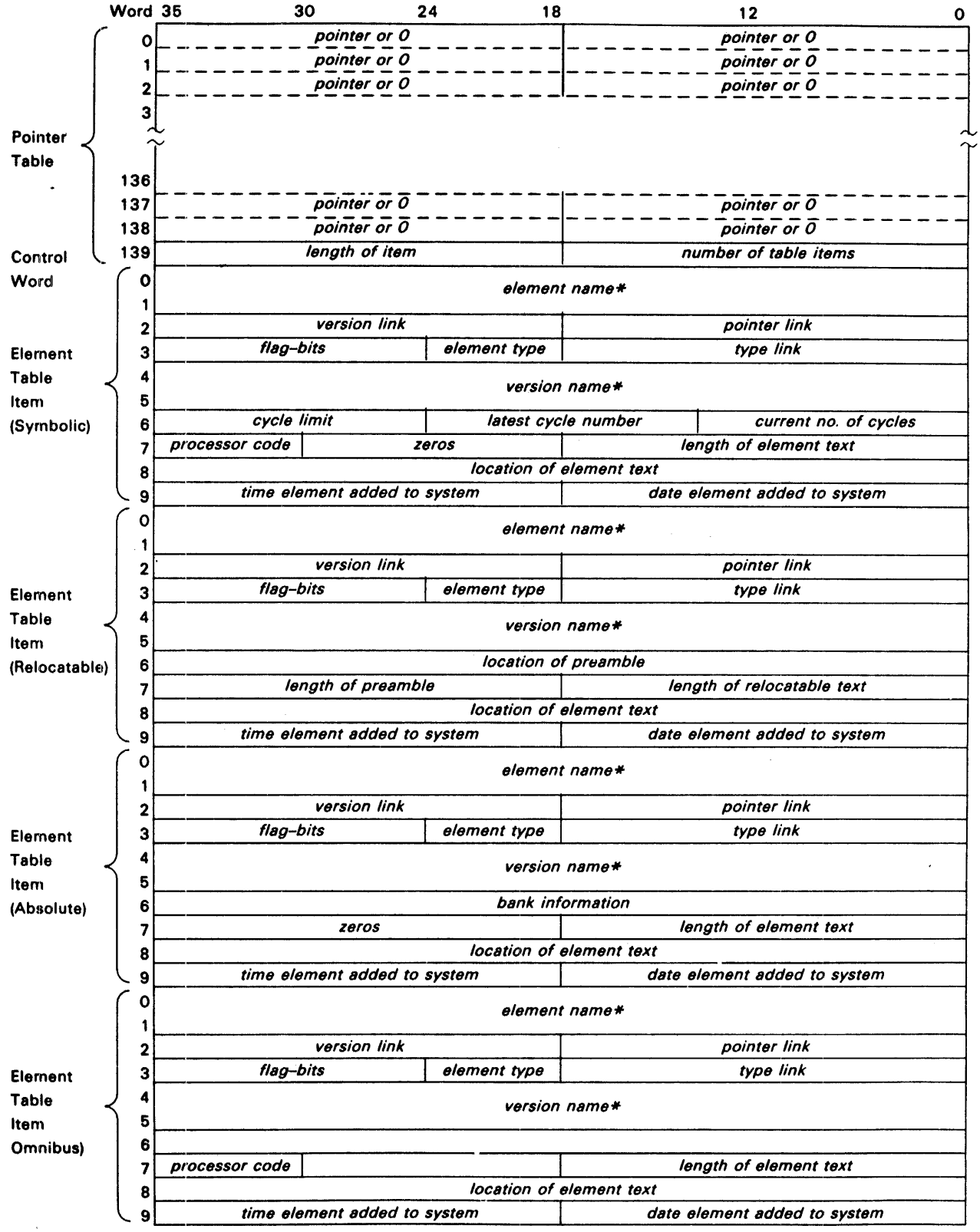


Figure 11-3. Element Table Format

The element table items follow the pointer table and appear in the order the elements were added to the program file.

Element chains are formed and linked by the use of pointers within the table entries themselves when two or more elements need to be pointed to from the same pointer half-word.

Version Link – Sequence number of another element item with the same element name and type, but a different version.

Pointer Link – Sequence number of another element item with the same hash code, but a different name.

Flag Bits – Bit 35 - Marked for deletion
31 - CTS flag (SYM)
30 - Arithmetic fault non-interrupt mode (ABS)
29 - Arithmetic fault compatibility mode (ABS)
28 - ASCII Code (SYM) or real-time (ABS)
26 - Third-Word sensitive (ABS or REL)
25 - Quarter-Word sensitive (ABS or REL)
24 - Marked in error (ABS or REL)

Element Type – The element types are:

SYM - Symbolic element	(type 01)
ASMP- ASSEMBLER procedure element	(type 02)
COBP - COBOL procedure element	(type 03)
FORP - FORTRAN procedure element	(type 04)

Types 2, 3, and 4 are also symbolic elements in structure and content. Their corresponding table items are identical to the Symbolic element table item (type 01).

REL - Relocatable element	(type 05)
ABS - Absolute element	(type 06)
OMN - Omnibus element	(type 07)

Element Subtype – See Volume 4-2.1.6 (SSTYP$) for the presently defined Omnibus and Symbolic element subtypes.

Type Link – Sequence number of another element item with the same element name, but with a different type.

Bank Information – Element Table Item Absolute word 6 contains the following information:

Word 9: Time and date information is in reversed TDATE$ format:

H1 = Time element created in seconds from midnight
H2 = Month/day/year element created

Word 6

Absolute Element

Word	S1	S2	T2	T3
.6	*flag bits*	*0*	*number of user banks*	*number of common banks*

Flag Bits – Bit 35 – Always set.
33 – Requires two PSRs due to initial and utility basing.
32 – Suppress zero fill.
31 – No start address for program.

11.2.1.2. Procedure Tables

The procedure table has an entry for each procedure (entered in the file by the @PDP control statement, see Section 8). Each entry consists of

- the procedure name,
- a link to the element in which it appears, and
- the procedure's relative word location within the file.

Each procedure table, like the element table, contains a 139-word pointer table and a control word.

All procedure table items contain an Element Link, a Pointer Link, and the location of the procedure within the file as well as the Procedure Name.

The Pointer Link is the sequence number of another item in the same procedure table with a different name, but the same hash code. It may be zero.

The Element Link is the sequence number of the element table item associated with the procedure element containing the procedure.

The location of the procedure is relative to the beginning of the file.

Bit 35 of word 3 of each procedure table item is the delete flag. If set, that procedure has been deleted, either by a delete request or by the insertion of a new procedure with the identical name (replacement of the old procedure).

Bit 34 of word three is the continuation indicator, when set, indicates that a second four words were necessary to contain the COBOL Procedure Name.

Bit 33 of word 3 indicates that the procedure images are in ASCII.

Bit 32 of word 3 indicates that the images contain sequence numbers in columns 73-80 which should be ignored.

The Assembler and FORTRAN procedure entries are as shown in Figure 11-4.

The COBOL procedure table item (see Figure 11-5) is either four or eight words long. If the procedure name is 12 alphanumeric characters or less, the item will be four words long. If the procedure name exceeds 12 characters (the system limits the name to 30), a second four words are used to complete the item. Bit 34 of word 3, when set, indicates that a second four words were necessary to contain the COBOL Procedure Name. Unused spaces in the name field will be Fieldata blank filled.

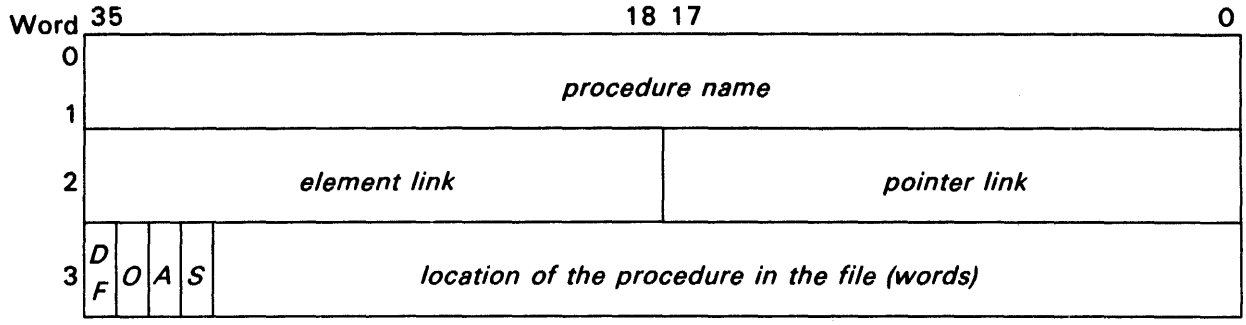


Figure 11-4. Assembler or FORTRAN Procedure Table Item

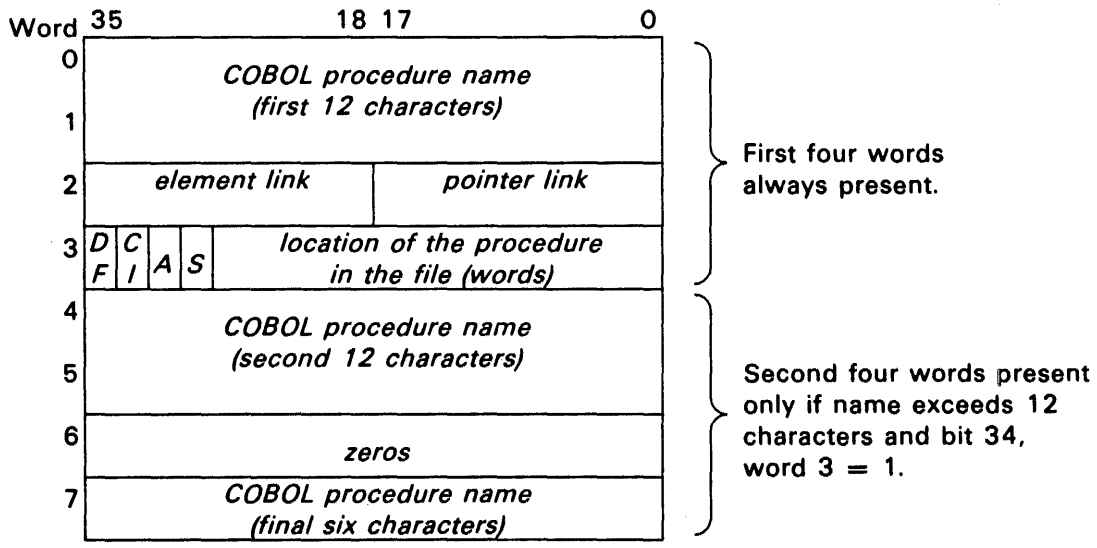


Figure 11-5. COBOL Procedure Table Item

11.2.1.3. Entry Point Table

The entry point table is the set of all entry point names and the link from each name to the relocatable element in which it occurs. The user must request the generation of this information using the @PREP control statement (see 4.2.11); it is not done automatically by the Executive.

The entry point table contains a 139-word pointer table and a control word, as did the element table.

The entry point table item is shown in Figure 11-6.

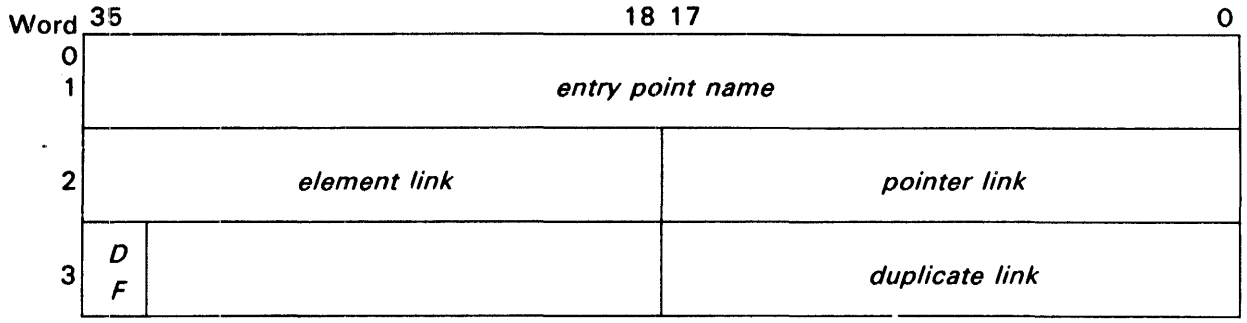


Figure 11-6. Entry Point Table Item

11.2.2. Element File Format

An element file is produced from a program file by using a @COPOUT control statement (see 4.2.3). It is a sequential file, found only on magnetic tape, and it may consist of a series of symbolic, relocatable, absolute, and omnibus elements. It may be temporary or a catalogued file.

The elements (see Figure 11-7) are written in sequential order on the tape. Each element (see Figure 11-8) contains a 28-word element label block and the element text. The element label block is created from information contained in the program file element table item and contains the following information:

- element file identifier
- element name, version, type, and size
- time and date the element was added to the file

The remainder of the element consists of 224-word blocks of the text of the element. This information is identical to the element text in the program file from which the element file was created. The only difference is that the element text is blocked into 224-word blocks; the last block is padded to force a 224-word block if the text does not occupy an exact multiple of 224 words.

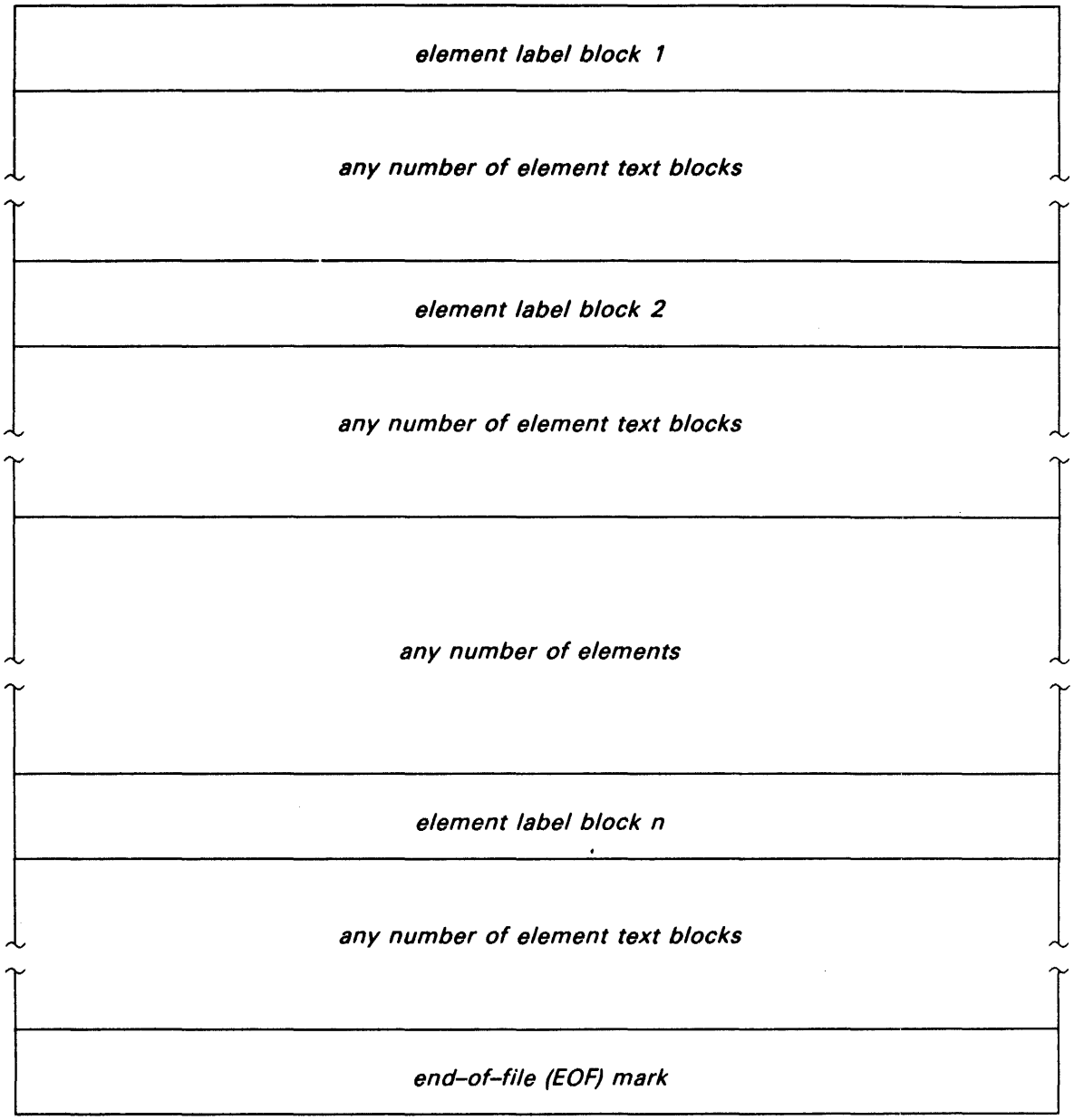


Figure 11-7. Element File Format

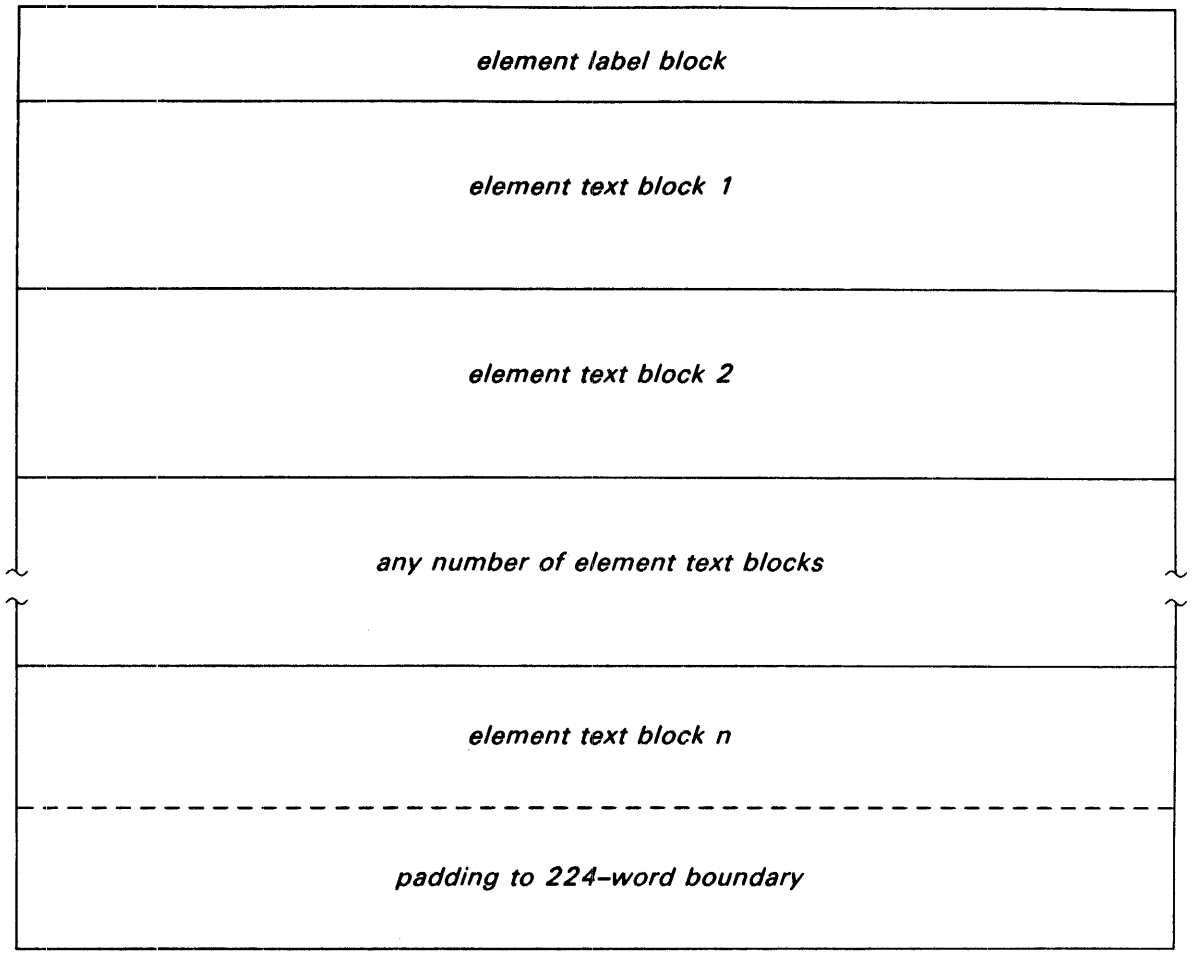


Figure 11-8. Element in Element File Format

11.2.3. System Data Format (SDF)

System Data Format (SDF) provides the system with a standard format for data handling between the various system components and between the system and the user. SDF files are produced by the @DATA processor, @FILE, the symbiont print and punch routines, the input symbionts, the FORTRAN library, the @ED processor, etc. SDF is also used as the format for symbolic elements in a program file.

SDF files on mass storage are a continuous set of sequential data. SDF files on tape are normally written in 224 word blocks. Images are allowed to span blocks except in symbiont output files.

Data is recorded in variable-length images with each image preceded by a control word containing the length of the image. There are two general types of control words (images):

1. Control images

   Control images provide special control information as needed by the system components processing the file or element. A control image is defined as one in which bit 35 is set in the control word. The control word contains a code in the range 040–077 in S1 of the control word. If an image follows the control word, its length is contained in S2. If no image follows, the content of S2 will be zero. The maximum length of a control image that may be defined with one control word is 63 words.

2. Data images

   A data image is defined as any image whose control word does not have bit 35 set. The remaining portion of T1 of the control word contains the length of the image. A maximum image length of 2047 words may be defined with one control word. The contents of the remaining portion of the control word varies depending on the specific type of SDF file or element.

11.2.3.1. Control Word Format for Control Images

The control word format is:

S1	S2	S3	S4	S5	S6
control code	*image length*	*SDF type*	*BRKPT part number*		*code type*

control code — A code within the range 040–077 that provides specific information about the SDF file or element. The currently defined control codes (octal values) are:

040 – Bypass image

Indicates that this image is to be skipped. S2 contains the number of words to skip. Skip to the next control word.

041 – Unique READ$ file label image

042 – ASCII/Fieldata switch

Used when an SDF file or element contains both ASCII and Fieldata images. Indicates a switch to the code type in S6 (ASCII = 1, Fieldata = 0) S2 is always 0.

043 – FORTRAN backspace

S3 contains the length of the previous image.

050 – SDF label

This is the initial control word of an SDF file or element. If a label image follows its length is in S2 and the image is always in

Fieldata even though the file or element may be in ASCII. S3 specifies the type of file or element. S6 specifies the code type of the following images in the file or element as does the 042 control image.

051 – Continuation

Indicates the following image is a continuation of the previous image. S2 specifies the number of words in this section of the image. This is used in symbiont files at the start of a 224-word block when an image spans blocks.

052 – Correction image

If H2 is nonzero, then H2 = the number of images deleted before the next image in the last update. If H2 = 0, this is a range or line correction statement whose character type is the same as the current SDF input. (see 4-2.1.4)

053 – H2 contains a CTS/HVTS line number, S2 and S3 are zero.

054 – End-of-reel

Used for multi-reel tape files to indicate a tape swap is required at the end of reel.

056 – Skip Break

Used so that information is not ignored when skipping forward in a print file.

060 – Print Control

070 – Punch Control

Used in symbiont print and punch files. S2 contains the number of words (image) which comprise the image submitted to the ERs PRTCH$/PCHCN$ or their ASCII and alternate equivalents. This image is then interpreted when the file is output to determine any mode change required. For example, an ER PRTCN$ to change the line width to 160 characters (27 words) would produce the image 600100000000_8 050534270505_8.

076 – Demand breakpoint EOF

The 076 end-of-file is used for intermediate EOFs such as breakpoints.

077 – End-of-file

Indicates termination of the SDF file or element.

image length — The length in words of the following image.

SDF type — A Fieldata character identifying the type or origin of the SDF file or element. Used only in label control words (CC = 041, 050). The types currently defined are:

00 – Unspecified SDF file/element type.

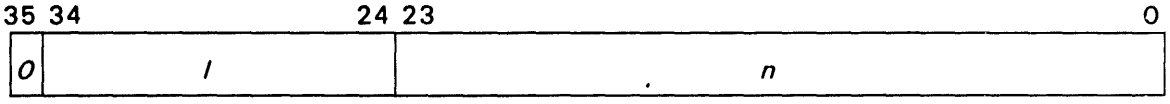
C – Symbiont card input/punch file
F – FORTRAN library data file
I – Symbiont input file (created by @FILE control statement)
P – Symbiont print file
S – Symbolic element. Usually created by the processor interface routine SIR$.
Used to indicate that bits 23–0 of data image control words contain element cycle information.
T – Symbiont paper tape input/punch file.

BRKPT part number — Used only in the label control word of symbiont files. Indicates the part number of the file, the count of the number of breakpoints performed on the file.

code type — The code type of the following images in the SDF file or element (Fieldata = 0, ASCII = 1) applies to control codes 042 and 050 only. Also applies to symbiont data control words.

11.2.3.2. Control Word Format for Data Images

The control word format is:



where:

l – The length in words of the following image. A maximum image length of 2047 may be specified.

n – The information contained in bits 23–0 varies with the type of SDF file or element. For symbolic elements this field contains element cycle information. For symbiont files this field contains line spacing and code type information.

11.2.3.3. Control and Data Image Formats

The formats for the control and data images of Processor Common I/O System (PCIOS) files are described in SPERRY UNIVAC 1100 Series Processor Common Input/Output System, UP-8478.

The label control image for SDF files and symbolic elements generated by the processor interface routine SIR$ is:

S1	S2	S3	S4	S5	S6
050	*001*	*'S'*			*code type*
SDFF					

The symbolic element data image control word format is:

35	34 — 24	S3	S4	S5	S6 — 0
0	*image length*	*numdle*	*dc*	*newc*	*ac*

where:

numdle – flag to indicate that images were deleted before this image on the last update.
dc – cycle at which this image was deleted
newc – set if image was added on this update
ac – cycle at which this image was added

The symbiont file label control image format is:

Word	S1	S2	S3	S4	S5	S6
0	*050*	*011*	*'P' or 'C'*	*part number*		*code type*
1	*filename (output) READ$X run-id (input)*					
2						
3	*device association*					
4	*run-id*					
5	*date and time of file creation*					
6	*user-id*					
7						
8	*reserved for 'SV' and 'SR' keyin use*					
9						

Words 1-2

User defined file – filename in Fieldata code

Input file - READ$xrunid
Output (print) file - PR@xxxrunid
Output (punch) file - PU@xxxrunid
where 'xxx' is the part number for the file.

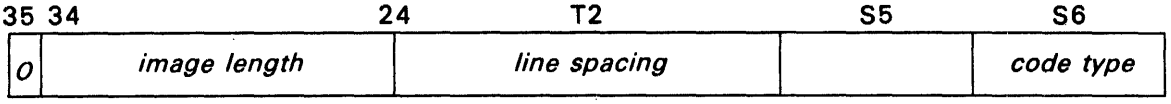
Word 3

Contains the input device name in Fieldata code for input and output files except for @SYM when it is the name of the output device specified on the @SYM statement.

Word 5

Date and time of the files creation, in binary code (TDATE$ format)

The symbiont data image control word format is:



The symbiont input file (@FILE) label control image format is:

050	001	'I'	
SDFF			

Examples of SDF control images:

1. Standard print file label block:

```
0 - 501125000000      -  standard print file label of 11(8) words.
1 - 252700606060      -  filename PR@, part number 000.
2 - 060606060606      -  run-id 'AAAAAA'.
3 - 102761050505      -  input device 'CR1'.
4 - 060606060606      -  run-id 'AAAAAA'.
5 - 010112005412      -  Jan 1, 1974 05412 seconds since midnight (ER TDATE$
6 - 050505050505         format).
7 - 050505050505      -  four words space filled.
8 - 050505050505
9 - 050505050505
10 - 000200010000     -  next SDF control for 2 word data image, space of 1.
11 - 002732220505     -  @RUN
12 - 060606060606     -  'AAAAAA'
          .
          .
          .
        etc.
```

2. Switch to ASCII

```
420000000001          -  the following images are ASCII until another control is
                         encountered.
```

11.3. FILE MAINTENANCE

Within the operating system are contained various library routines, Executive service functions, and processors that may be used to create and manipulate files. The file utility processor, FURPUR (see Section 4) may be used to process, in various ways, files of the previously discussed formats. In fact, element files are created and processed only by the FURPUR processor.

SDF files and elements are produced and processed by the FURPUR, DATA, ED, ELT, and other processors. In addition, the symbiont and @ADD control statement (see Volume 2-3.10.1) as well as the various symbiont interface Executive functions (see Volume 2-Section 5) are used to create and process SDF.

Program files are created and processed by the language processors, FURPUR, ELT, LIST, CULL and other processors. In addition, there are several Executive service functions (see 11.3.1) and relocatable library routines (see BSP$) available for processing program files.

Paragraphs 11.3.1 and Volume 4 - BSP$ describe the mechanism for updating a program file by a user program. Both features were designed primarily for use by language and system processors. The program file maintenance Executive Requests (see 11.3.1) provide a limited capabilty in that only selected functions are available. The program file Basic Service Package BSP$ (see Volume 4) is a system relocatable library routine that can be included with a user program to provide more capability with less overhead if many operations are to be done.

The Executive Requests are also used by the Executive in its normal operations, such as finding an absolute program to execute. See 11.3.1.6 for program file package status code.

11.3.1. Program File Maintenance Executive Requests

The Executive Requests described in the following paragraphs are used to maintain the table of contents entries for a program file. As a group, the requests are called the program file package (PFP). The formats of the program file table of contents entries are described in 11.2.

For each of the requests described, upon return from the requests, register A2 contains the status of the operation performed. Entries can be found in 11.2.1.

PFP packets cannot be in write protected banks.

11.3.1.1. Updating the Element Table (PFI$)

Purpose:

Inserts an entry in the program file's element table.

Formats:

```
L,U     A0,pktaddr
ER      PFI$
```

or

```
L       A1,next write location
LN,U    A0,pktaddr
ER      PFI$
```

The second calling sequence combines functions of ER PFI$ and ER PFUWL$.

Pktaddr is the address of a packet whose format is:

Word	T1	S3	H1
0	internal filename		
1			
2	element-name		
3			
4	*version-link-sequence-nbr*		*pointer-link-sequence-nbr*
5	flag-bits	element-type	*type-link-sequence-nbr*
6	element-version-name		
7			
8			
9			
10	text-address		
11	date-and-time-of-creation		

Words 2-11 are the same as the contents of the Element Table Items. (See Figure 11-3 and the associated description.) Note that the contents of words 8 and 9 vary with the type of element.

Description:

The element version can be present, zero, or blank. When the version is zero, blanks are substituted.

When an absolute element is being inserted, its sequence number is recorded in the file table index.

For the relocatable elements, the file table index pointers to the entry point table are cleared and a new entry point table may have to be created.

If the date-and-time-of-creation field is zero, the PFI$ request inserts the current data and time. When this field is nonzero, the contents are used for the date and time.

The link sequence numbers are supplied by the PFI$ request.

11.3.1.2. Table of Contents Search (PFS$)

Purpose:

Searches program file's table of contents for a given item.

Format:

```
L,U  A0,pktaddr
ER   PFS$
```

Pktaddr is the address of a packet whose format is identical to that of the PFI$ request (see 11.3.1.1).

Words 8 and 9 of the packet are suppled by the Executive.

Description:

When the delete flag (bit 35 of word 5) is set, a find can be made on an element marked for deletion.

When the element name is left blank and the element desired is an absolute element, the PFS$ request supplies the last absolute element added to the file.

If the version name is zero, a find is made on element name only. When a version name other than zero is specified, it is used along with the element name in the search. When a version name is blank, a find is justified on the element name and blanks, for the version.

When the element is found, the packet is filled with information from the element table entry, and the sequence number of the element is returned in A1. This information is used to access the element text.

11.3.1.3. Mark Element for Deletion (PFD$)

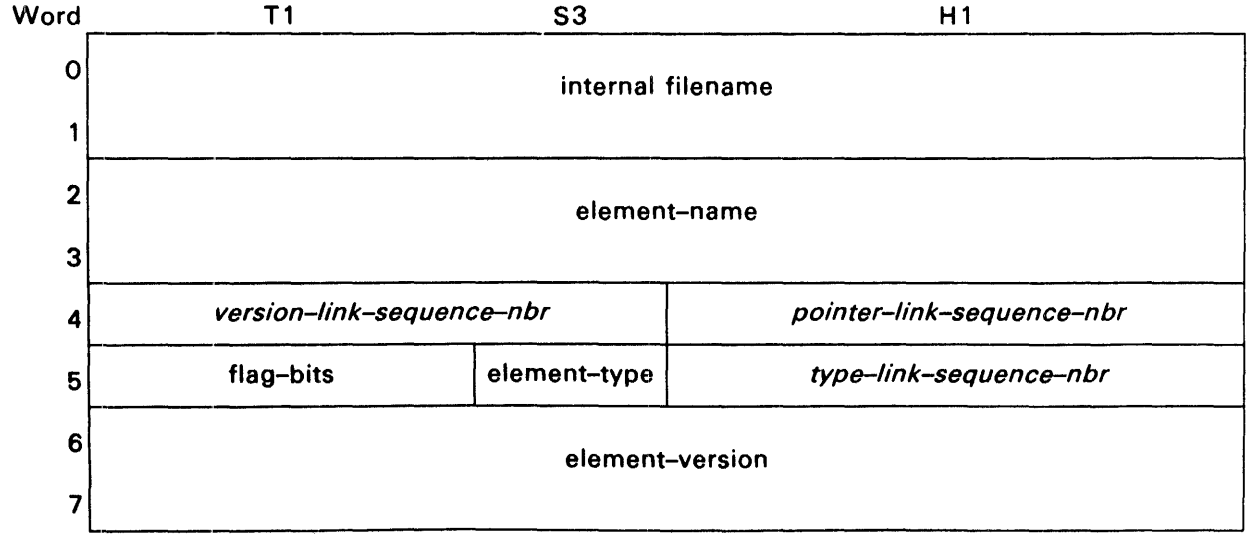
Purpose:

Sets delete flag in element table item for requested element.

Format:

```
L,U  A0,pktaddr
ER   PFD$
```

Pktaddr is the address of a packet whose format is:



Word 5

element-type Same as PFI$ (see 11.3.1.1).

Description:

When the element being deleted is the most recently added absolute element in this file, the file table index entry containing the element sequence number of the most recently added absolute element is cleared to zero.

11.3.1.4. Updating Next Write Location (PFUWL$)

Purpose:

Updates the next write location in a program file. The task of the PFUWL$ request may also be performed by using PFI$ (see 11.3.1.1). This is accomplished by complementing register A0 on the call to PFI$ and loading register A1 with the new address.

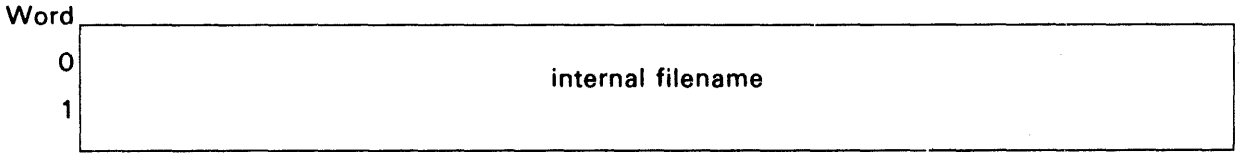
Formats:

```
L,U  A0,pktaddr
L    A1,(new-address-in-program-file)
ER   PFUWL$
```

or

```
LN,U A0,pktaddr-for-PFI$
L    A1,(new-addr-in-program-file)
ER   PFI$
```

Pktaddr is the address of a packet whose format is:



The next write location is the next available sector at which the text portion of the element can be written without destroying other text words.

11.3.1.5. Retrieving Next Write Location Address (PFWL$)

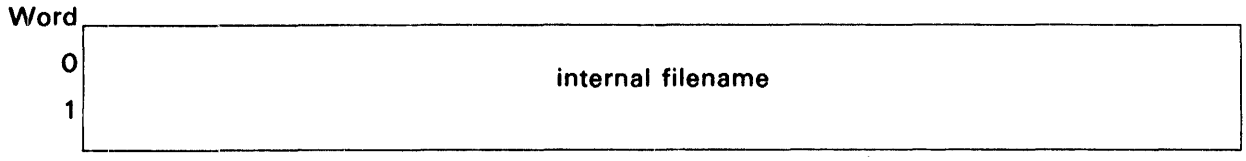
Purpose:

Obtains the next write location in the program file.

Format:

```
L,U  A0,pktaddr
ER   PFWL$
```

Pktaddr is the address of a packet whose format is:



The next write location is stored in register A1 upon normal return. The next write location is as defined for PFUWL$ (see 11.3.1.4).

11.3.1.6. Program File Package Status Conditions

The status conditions are stored in register A2 on the return from the program file package ERs. The possible values are:

00 – Normal status (operation completed)

01 – No find

02 – I/O error, or file not assigned

03 – Program file not defined as program file

04 – reserved

05 – PFI$ request reject due to packet address wholly or partially outside the I–bank or D–bank limits.

06 – Packet wholly or partially outside I– or D–bank limits

07 – Sequence number greater than 5001 encountered

Appendix A. Collector Diagnostic Messages

ABOVE F-CYCLE SUBFIELD NOT PROPERLY FORMATTED

The F-cycle subfield in the immediately preceding source statement is improperly formatted.

ABOVE FILE IS READ ONLY - COLLECTOR CANNOT OUTPUT INTO IT

ABOVE READ KEY SUBFIELD NOT PROPERLY FORMATTED

The read key subfield in the immediately preceding source statement is improperly formatted.

ABOVE STATEMENT NOT PROCESSED DUE TO FORMAT ERROR

ABOVE STATEMENT NOT PROCESSED. IF BANKS ARE DESIRED THE FIRST SOURCE IMAGE MUST BE A BANK STMT

An I-BANK, D-BANK, or FORM statement was encountered after a SEG, RESG, DSEG, XSEG or IN statement has been processed.

ABOVE STMT NOT ALLOWED FOLLOWING A COMMON BANK

Following an I-BANK or D-BANK statement specifying a common bank, another I-BANK or D-BANK statement must precede any SEG, DSEG, RSEG, XSEG or IN statement.

A COMMON BANK CAN HAVE NO RELATIONSHIPS

An I-BANK or D-BANK statement with an X option has a bank-list subfield specified.

I-BANK or D-BANK ALREADY BASED ON MAIN or UTIL PSR

The M or U option has been specified on more than one I-BANK or D-BANK statement.

AFCM STATUS OF OUTPUT ELEMENT = CLRAFCM

The Arithmetic Fault Compatibility Mode for the output element is set as clear. An initial load on the 1110 or 1100/40, the Arithmetic Fault compatibility mode setting will be such that no interrupt occurs for floating point overflow, floating point underflow and divide fault. (See 2.2.2.13)

AFCM STATUS OF OUTPUT ELEMENT = INSAFCM

The Arithmetic Fault Compatibility Mode for the output element is set as insensitive. An initial load on the 1110 or 1100/40, the Arithmetic Fault Compatibility Mode setting will be such that no interrupt will take place in the case of floating point overflow, floating point underflow and divide fault. (See 2.2.2.13)

AFCM STATUS OF OUTPUT ELEMENT = SETAFCM

The Arithmetic Fault Compatibility Mode for the output element is set as set. At initial load on the 1110 or 1100/40, the Arithmetic Fault Compatibility Mode will be such that an interrupt will occur for floating point overflow, floating point underflow, and divide fault. (See 2.2.2.13)

AFCM STATUS OF OUTPUT ELEMENT = UNKNOWN

The Arithmetic Fault Compatibility Mode for the output element is set as unknown. At initial load on the 1110 or 1100/40, the Arithmetic Fault Compatibility Mode will be determined by a system generation parameter. (See 2.2.2.13)

ASSIGN ABOVE FILE LARGER MAX SIZE

Value AT ADDRESS address BITS right-most – left-most IN ELEMENT element name IS POSSIBLE OVER-65K ADDRESS FIELD.

Instruction AT ADDRESS address IN ELEMENT element name – WILL ACTIVATE SNAP JUMP PRIOR TO EXECUTION

A snapshot will be taken at the specified address and then the specified instruction will be executed.

BAD PLACEMENT OF CHARACTER x

BAD STATUS: xxxxxxxxxxxx OUTPUT BY CSF$ IN ATTEMPT TO @ASG,AX filename

A dynamic @ASG,AX of the file resulted in a bad status return (See Volume 2–C.2 for explanation of 12 digit (x...x) octal code.)

BAD STATUS: xxxxxxxxxxxx OUTPUT BY CSF$ IN ATTEMPT TO @FREE,A filename

A dynamic @FREE,A of the file resulted in a bad status return. (See Volume 2–C.2 for explanation of 12 digit (x...) octal code.)

BAD STATUS: xxxxxxxxxxxx OUTPUT BY CSF$ IN ATTEMPT TO @USE filename

A dynamic @USE of the file resulted in a bad status return. (See Volume 2–C.2 for explanation of 12 digit (x...x) octal code.)

BANK INDIRECTLY DEFINES ITSELF

Some bank in the bank structure is related to a bank that is directly or indirectly related to the first bank. For example, bank A follows bank B and bank B follows bank A.

BANK IS NOT PROPERLY DEFINED: bank name

The bank has not been specified in the name–1 subfield on an I–BANK or D–BANK statement.

BANK, SEGMENT OR SEG WITHIN BANK NOT PREVIOUSLY DEFINED

A FORM has been done or an undefined segment or bank, or on a segment that is not contained in the specified bank.

BANK STMTS NOT ALLOWED IN R-OPTION COLLECTION

BOTH M AND U OPTIONS ON BANK STATEMENT - option IGNORED

Both the M and U options are specified on the immediately preceding I-BANK or D-BANK statement. The second option is ignored.

BOTH T AND F OPTIONS GIVEN - BOTH IGNORED

The @MAP control statement contained both the T and F options.

CLRAFCM = SENSITIVITY OF ABSOLUTE ELEMENT/SETAFCM = SENSITIVITY OF START ADDR ELEMENT

When the program is loaded for execution on the 1110 or 1100/40, the arithmetic fault compatibility mode will be set such that no interrupt is taken when a floating point overflow, floating point underflow or divide fault occurs. However, when execution is initialized, the program's code expects interrupts to be taken. This is a logical conflict.

COLLECTOR DIAGNOSTIC MESSAGES

CONTROL BANK AMBIGUITY - C OPTION IGNORED

The C option may be specified once only in a collector on either an I-BANK or D-BANK source statement. The C option had already been specified once.

COMBINED LC lc no. LENGTH EXCEEDS 65K - NON FATAL ERROR

This message occurs with an R option collection when the combined lengths of all location counters of the specified number in the included relocatable elements exceeds 65K.

CONTROL BANK HAS S OPTION - SEGMENTATION NOT ALLOWED

A program whose control bank has the S option specified contains segments.

COMMON BANK CANNOT BE CONTROL BANK

Both C and X options have been specified on the same I-BANK or D-BANK source statement. The C option (control bank) is ignored.

CORRECTION CARD SEQUENCE ERROR

COR: patch REPLACED original text AT ADDRESS address IN ELEMENT element name

COR CALLS ON UNDEFINED LC lc no. IN ELEMENT element name

The specified LC no. is not present in the named relocatable element.

COR OF NON-EXISTENT ELEMENT

COR NOT USED WITH R-OPTION

D-BANK bankname ASSIGNED SUCH THAT SETMIN value IS SATISFIED

The start address of the named bank has been adjusted so that the specified SETMIN value has been satisfied. The start addresses of all other banks related to this bank are determined by the adjusted start address.

DEF AND *REF* CANNOT BOTH HAVE NAME: entry point

The same entry point name has been specified on both a DEF and REF source statement. The first source statement with the name is the only one processed. All others are ignored.

DEF and *REF* NOT USED WITH V-OPT

A DEF or REF statement has been specified with a V option @MAP. The statements are ignored because the tables generated by these statements are placed in the program's D-bank which will be non-existent in the absolute element.

DIRECTED LIB FOR FILE filename ILLEGAL - NONDIRECTED ASSUMED

A directed LIB statement has been found in a bank-implied collection. The statement is processed as simply LIB filename.

bank name DOES NOT EXIST AS A COMMON BANK

The name common bank specified for initial basing is not defined to the Executive System as a common bank.

bank name DUPLICATELY DEFINED

The specified bank name has already been encountered on an I-BANK or D-BANK statement. This results in a fatal error, i.e., no absolute element is produced.

element name ELEMENT AMBIGUITY IN FILE filename

The specified element name is found for more than one relocatable element in the file. The list following the message contains the element name and version name for each such element in the file. None of the listed elements is included in the collection.

ELEMENT element name HAS MAP SPECIFIED LCS GREATER THAN MAX LC no. CONTAINED IN RB

A location counter number has been specified on an IN statement or an $lcs statement which is greater than the largest LC no. contained in the element.

ENT ENTRY POINT name NOT GLOBAL - NOT USED

The named entry point which was specified on an ENT statement is not global.

ENT SPECIFIED ENTRY POINT IS NOT IN ANY INCLUDED ELEMENT - NOT USED: entry point name

ENT SPECIFIED ENTRY POINT IS NOT IN INITIALLY BASED BANK - NOT USED: entry point name

The program start address must be in an initially based bank.

ENT SPECIFIED ENTRY POINT IS NOT IN THE MAIN SEGMENT – NOT USED: entry point name

The program start address must be in the main segment.

ENT STATEMENT CANNOT HAVE A NUMERIC FIELD

The parameter field of the ENT statement must contain an externalized entry point name

element name ENTRY POINT entry point name ALREADY DEFINED

The specified entry point found in the named element has already been defined in another element. The entry point in the named element is not used.

element name ENTRY POINT entry point name ALREADY DEFINED BANK STATEMENT

element name ENTRY POINT entry point name ALREADY DEFINED BY EQU STATEMENT

element name ENTRY POINT entry point name ALREADY DEFINED BY REF STATEMENT

element name ENTRY POINT entry point name ALREADY DEFINED BY SEG STATEMENT

entry point name ENTRY POINT AMBIGUITY IN FILE filename

entry point name ENTRY POINT NOT FOUND FOR COR

entry point name ENTRY POINT NOT FOUND FOR SWAP

ENTRY POINT entry point name – USED TO ACTIVATE INDIRECT SEGMENT LOAD – IS ILLEGALLY REFERENCED WITH A PLUS OR MINUS OFFSET FROM OUTSIDE OF ITS SEGMENT FROM ELEMENT element name

An entry point in the I-BANK of an indirectly loaded segment cannot be referenced with a plus or minus offset unless the reference is made from within the segment containing the entry point. If the reference is from outside the segment containing the entry point, the offset is ignored.

EP entry point name NOT GLOBAL – 0 USED IN COR

A value of zero has been used for an entry point specified in a COR source statement as no global value was found for the entry point.

EP entry point name NOT GLOBAL – 0 USED IN SNAP

A value of zero has been used for an entry point specified in a SNAP source statement as no global value was found for the entry point.

ERROR IN ELEMENT: element name

The specified element was marked in error by the processor that generated it.

FATAL ERROR

ELEM element name FOR RSEG seg name CANNOT LOAD DATA INTO COMMON BLOCK common block name LOCATED IN SEG seg name

**FATAL ERROR: I-BANK ADDRESS EXCEEDS 0177777 (65K DECIMAL)

The I-BANK address in a bank-implied collection exceeds 65K decimal.

**FATAL ERROR: I-BANK bank name ADDRESS EXCEEDS 0177777 (65K DECIMAL)

The specified bank in a bank-named collection has addresses exceeding 65K decimal.

**FATAL ERROR - NO OUTPUT ELEMENT PRODUCED BY COLLECTOR

A fatal error in the collection has prevented the output of an absolute element. The preceding diagnostic messages will specify the error or errors.

FATAL ERROR NO RB ELEMENT PRODUCED

Due to a fatal error, no relocatable element was produced in an R option collection. The preceding messages will specify the error or errors.

FATAL ERROR - NO RB ELEMENT PRODUCED - HIGHEST LC no. ALLOWED IS 077 - HIGHEST LC no. NEEDED IS lc no.

In an R option collection, all common blocks are assigned location counter numbers greater than the highest location counter number found in any of the included relocatable elements. This message occurs when there are not enough location counter numbers available for assignment to all common blocks.

**FATAL ERROR: PROGRAM IS TOO BIG - ADDRESSES OVER 0777777 ARE TRUNCATED

In a bank-implied collection, the assigned D-bank addresses exceed 262K decimal.

**FATAL ERROR: PROGRAM IS TOO BIG - ADDRESSES OVER 0777777 ARE TRUNCATED FOR D-BANK bank name

In a bank-named collection, the assigned addresses for the named D-bank exceed 262K decimal.

FILE filename NEEDS A PREP IN ORDER TO BE SEARCHED

The named file specified on a LIB statement has not been @PREPed. The file is not searched.

FIRST SEGMENT IS MAIN SEGMENT - MAY NOT BE DSEG

The first segment named in a bank-implied collection and the first segment named following a bank statement in a bank-named collection cannot be a dynamic segment.

FIRST SEGMENT IS MAIN SEGMENT - MAY NOT BE RSEG

The first segment named in a bank-implied collection and the first segment named following a bank statement in a bank-named collection cannot be a relocatable segment.

FORM ON BANK-NAME NOT ALLOWED AFTER SEG OR IN STMT

bank name HAS ALREADY BEEN USED AS NON-BANKNAME

A bank name must be unique from all segment names and entry points in the collection.

type parameter IGNORED – FORMAT OR TERMINATOR ERROR

The TYPE statement has an illegal terminator following the specified parameter or the parameter is illegal.

type parameter IGNORED – TYPE PARAMETER CONFLICT

The named parameter directly contradicts a parameter already processed on the TYPE parameter.

ILLEGAL GROUP NO no. IN INFO DIRECTIVE IN ELEMENT element name IS IGNORED

If the INFO directive had an entry point attached to it, group no 2 specifying a named common block is assumed. If no entry point was attached to the INFO directive, group no 4 specifying blank common is assumed.

ILLEGAL OPTION option IGNORED

The named illegal option was specified on an I–BANK or D–BANK source statement. The option is ignored and the collection is continued.

element/version IN filename BYPASSED – DUPLICATED ELEMENT NAME ALREADY SPECIFIED

In processing a whole file IN (IN FILENAME.), an element in FILENAME is found which duplicates the name of an element IN'd from another file.

value IS ILLEGAL LENGTH FOR LOCATION COUNTER no. IN ELEMENT element name

The relocatable element's preamble specifies a length of greater than 65K decimal for the given LC no. This indicates that the RB was incorrectly generated or that the file containing the RB has been wholly or partially destroyed.

entry point name IS NOT DEFINED – REFERENCED IN ELEMENT element name

LC no. UNKNOWN TO LC no. ELEMENT element name BANK bank name

Due to local inclusion of the element, the reference made to the first LC no. cannot be satisfied as it is not in any bank–set specified for the second LC no.

LIB filename ( ) IGNORED – NO PREVIOUS LIB DIRECTION GIVEN

The named file is not searched as no LIB (BANK/$lcs,...) was given.

LIMIT = 16 SNAPS/COLLECTION

The immediately preceding SNAP statement is ignored as 16 snaps have already been made.

LOCAL–GLOBAL CONFLICT FOR EP entry point – REFERENCED BY LC no. ELEM element name BANK bank name

Due to local inclusion of the location counter under which the entry point is named, more than one value for that entry point can be referenced from the named bank. A value of zero is used to satisfy the reference to the entry point.

LOCAL-GLOBAL CONFLICT FOR LC no. - REFERENCED BY LC no. ELEM element name BANK bank name

The first location counter has been locally included such that the named bank can access the location counter in more than one bank. A value of zero is used for the LC's assigned value as it is impossible to determine which of the available values should be used.

MAIN SEG NAME MUST BE SAME FOR ALL BANKS

In a bank-named collection, the main segment of each bank must have the same name as the main segments of all other banks.

MAP TERMINATED DUE TO IMPROPER FORMAT OF BANK STMT

The immediately preceding I-BANK or D-BANK statement is improperly formatted.

element name MINIMUM ADDRESS IGNORED
BANK HAS USER SPECIFIED STARTING ADDRESS

LC 0 in the element is contained in a bank that has numeric start address specified by the user. The SETMIN for the element is ignored.

element name MINIMUM ADDRESS IGNORED - LC 0 NOT IN D-BANK

The SETMIN value for the element applies to LC 0. LC 0 must be contained in a D-BANK or else the SETMIN value is ignored.

MINIMUM GAP SIZE IN ERROR

SYSTEM VALUE 10 IS USED

A format error was detected on the MINGAP source statement. The statement is ignored and the system value is assumed.

MINIMUM LOAD SIZE IN ERROR
SYSTEM VALUE 10 IS USED

A format error was detected on the MINSIZ source statement. The statement is ignored and the system value is assumed.

MORE THAN ONE GLOBAL COPY OF A LOCATION COUNTER IS SPECIFIED FOR ELEMENT element name

In a bank named collection, the element or part of the element has been included more than once without a local bank-set list. An element or part of an element may be included locally many times but can only be included globally once.

MORE THAN 63 SEGMENTS NAMED ON SEG STMT

A maximum of 63 segments can be specified, either explicitly or implicitly, in the relationship list on a SEG statement. This causes a fatal error.

NO CONTINUATION STATEMENT FOUND

No further statement was found following the continuation character (;).

NO ELEMENTS IN SEGMENT: segment name

NO ELEMENTS IN SEGMENT: segment name, BANK; bank name

This message is produced in bank-named collections when a segment is found to have no elements included in it.

NO ELEMENT WITH *DEF* ENTRY POINT: entry point name

No element was found which contained the specified DEF entry point.

NO GLOBAL LC no IN ELEMENT element name - COR IGNORED

A COR source statement specified an LC that was not included globally in a bank-named collection.

NO GLOBAL LC no. IN ELEMENT element name - SNAP IGNORED

A SNAP source statement specified an LC that was not included globally in a bank-named collection.

element name NOT FOUND IN FILE filename

*NOT NEEDED - MAIN SEGMENT STAYS LOADED

The indirect load indicator (*) was used in specifying the main segment. The asterisk is ignored.

NO START ADDRESS

No ENT statement was used in the collection and none of the preambles of the included relocatable elements indicated a start address.

NUMBER IN COR ADDRESS FIELD IS OVER 0177777 (65K)

OFFSET NOT NUMERIC ON EQU STATEMENT

instruction POSSIBLE BAD INSTRUCTION AT address IN ELEMENT element name

$PREFIXED TO COMMON BLOCK NAME TO AVOID DUPLICATING ELEMENT NAME element name

PREVIOUS ENT STATEMENT OVERRIDES THIS ONE

PREVIOUS FORM ON BANK PRECLUDES DEFINING ANOTHER SEG FOR THIS BANK

Since a FORM on a bank-name creates an exact duplication of a previous bank structure, no additional SEG, DSEG, or RSEG statements may be specified for the bank being generated.

PREVIOUS IN OVERRIDES THIS ONE FOR FILE: filename

In a bank-implied collection, an IN of a whole file can occur once only. Only the first IN FILENAME is processed.

PREVIOUS IN OVERRIDES THIS ONE FOR THE ELEMENT NAME: element name

In a bank-implied collection, an element can be included once only in a collection. Only the first inclusion is processed.

PROG OVERFLOWS TABLE SPACE - ERROR IN COLLECTOR AT ADDR: address

The collector has used all of its available table space in an attempt to collect the program. An analysis of the dump is necessary to determine the reason for the table overflow.

RB ELEMENT NOT FOUND: element name

RB INPUT ELEMENT ERROR: element name - K VALUE value

A badly formatted relocatable element has been found. This usually indicates that either the relocatable was badly formatted when it was produced or that all or part of the file containing the element has been destroyed.

READ KEY NECESSARY WITH FILE: filename

The Collector cannot dynamically assign the file because no read key was specified with the filename.

entry point name REFERENCED IN ELEMENT element name NOT DEFINED FOR BANK bank name

Because of local element inclusion, the specified entry point cannot be referenced from the named bank.

RSEG CANNOT BE USED TO DEFINE A SEG

A relocatable segment cannot be present in the relationship field of a SEG statement. The RSEG is ignored in the relationship list.

RSEG STATEMENT USES NO FIELDS OR CHARACTERS OTHER THAN NAME

RSEG TOO LARGE - CANNOT EXCEED 65K

SAME LOCATION COUNTER USED FOR DIFFERENT COMMON BLOCKS IN ELEMENT element name

S OPTION BANKS DO NOT ALLOW SEGMENTS TREATED AS DYNAMIC

A SEG statement has appeared for a bank which was specified with an S option.

SEG CANNOT OVERLAY MAIN SEG. HAS BEEN CHANGED TO FOLLOW MAIN SEG

SEG NOT USED WITH R-OPTION

SEG NOT USED WITH V-OPTION

SEG NOT USED WITH Y-OPTION

SEGMENT INDIRECTLY DEFINES ITSELF

Some segment in the segment structure is related to a segment that is directly or indirectly related to the first segment. For example, segment A follows segment B and segment B follows segment A.

SEGMENT NAME DUPLICATED segment name

The specified name has been used to define more than one segment in the collection. This results in a fatal error.

SEGMENT NOT PROPERLY DEFINED: segment name

The segment name has not been specified in the name–1 subfield on a SEG or DSEG statement.

SETAFCM = SENSITIVITY OF ABSOLUTE ELEMENT/CLRAFCM = SENSITIVITY OF START ADDR ELEMENT

When the program is loaded for execution on the 1110 or 1100/40, the arithmetic fault compatibility mode will be set such that an interrupt is taken when a floating point overflow, floating point underflow, or divide fault occurs. However, when execution is initialized, the program's code does not expect an interrupt to occur.

SNAP CALLS ON UNDEFINED LC no. IN ELEMENT element name

The LC no. on a SNAP statement does not exist for the element.

SNAP$ ELEMENT NOT FOUND – NO SNAPSHOTS TAKEN

The SNAP$ element which produces the requested snaps has not been found, thus preventing any snapshots.

SNAP OF NON–EXISTENT ELEMENT

SNAP NOT USED WITH R–OPTION

START ADDR ALREADY FOUND – ONE NOT USED IN ELEMENT element name

START OF D–BANK SET ABOVE CURRENT STANDARD 040000 IN ORDER TO MEET MINIMUM ADDRESS REQUIREMENT value – COM BLOCK common block name

START OF D–BANK SET ABOVE CURRENT STANDARD 040000 IN ORDER TO MEET MINIMUM ADDRESS REQUIREMENT value OF ELEMENT element name

START OF D–BANK SET AS FAR AS NECESSARY OR POSSIBLE BELOW CURRENT STANDARD 040000 to MINIMIZE USING ADDRESSES OVER 0177777 – PROGRAM MAY NOT LINK$ SUCCESSFULLY TO REENTRANT PROCESSORS – ALTERNATIVE IS TO USE E OPTION

START ADDR OF ELEMENT NOT IN MAIN SEGMENT – NOT USED: element name

SYMBOL NAME symbol DUPLICATION ERROR

The specified name, present on an RSEG, SEG, DSEG, I–BANK or D–BANK statement, has already been found a non–segment or non–bank name. This produces a fatal error.

SYSTEM NOTE: STANDARD REENTRANT PROCESSOR MAY NOT EXCEED ADDRESS 037777

**THE TRUNCATION CHECK IS TURNED OFF AFTER 500 WARNINGS

More than 500 instructions have been found in which the address portion of the word has had to be truncated.

THIS FILE NOT CORRECTLY CATALOGUED OR ASSIGNED–NOT FOUND: filename

THIS SEGMENT IS USED TO DEFINE ITSELF

The segment being defined in the immediately preceding SEG or DSEG statement has its own name present in the relationship list. The name in the relationship list is ignored.

TOO MANY CHARACTERS IN A SUB-FIELD

A sub-field in the immediately preceding source statement has too many characters. The source statement is bypassed.

TPF$ TREATED AS AN ELEMENT

As IN TPF$ was encountered. As no period followed the name, the name is assumed to be that of an element. On the IN statement, all files must have a period immediately following the filename.

USER MAX TIME (PAGES) MET IN COLLECTION

V-OPTION NOT USED WITH R-OPTION

WARNING: EP entry point name UNDER VOID LC no. ASSIGNED VALUE value

An entry point which is assigned under a location counter with no length has been given the specified value.

WARNING: REFERENCE TO VOID LC no. IN RB ELEMENT element name SATISFIED WITH VALUE value

In an R option collection, a reference has been found to a location counter with no length. The specified value was used to satisfy the reference.

WARNING-TRUNCATION OF FIELD value AT ADDRESS address BITS right-most - left-most IN ELEMENT element name

The value to be placed in the specified field size is too large and was truncated. The left portion of the value is truncated. This occurs, for instance, when an over 65K address is supposed to be placed in the U portion of an instruction which is only 16 bits long.

WRONG PROJECT ID TO ACCESS PRIVATE FILE: filename

Y-OPTION NOT USED WITH R-OPT

CUT

USER COMMENT SHEET

Comments concerning the content, style, and usefulness of this manual may be made in the space provided below. Please fill in the requested information.

Requests for copies of manuals, lists of manuals, pricing information, etc. should be made through your 1100 Series site manager to your Sperry Univac representative or the Sperry Univac office serving your locality.

System: _____

Manual Title: _____

UP No: _____ Revision No: _____ Update: _____

Name of User: _____

Address of User: _____

Comments:

FOLD

FIRST CLASS

PERMIT NO. 21

BLUE BELL, PA.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

SPERRY UNIVAC

SYSTEMS SUPPORT
ATTN: INFORMATION SERVICES M.S. 4533
P.O. BOX 3942
ST. PAUL, MINNESOTA 55165

CUT

FOLD