SELECTIT

Table of contents

Introduction

Overview

SELECTIT is a utility which provides a wide variety of functions related to copying, printing and/or modifying datasets. Any and all SELECTIT functions can be combined in a single run or requested individually, as you desire.

SELECTIT can sequentially read up to 9 input files (sequential, partitioned, ISAM, or keyed VSAM) and selectively create any number of output files.

You may use SELECTIT to copy records (with or without alteration of the records) and/or print them.

Note: If pointed to in the JCL via DSN=dataset(member) then a member of a PDS is considered to be a sequential dataset. If the PDS is allocated without a member name, refer to "Processing PDS Datasets" for special considerations.

Features

- **File Copy** Copy a record from an input file to an output file (unconditionally).
- **Selective Copy** Copy a record only if your supplied condition is met.
- **Limit Copy** Copy a limited number of records (possibly in conjunction with a user supplied condition).
- **File print** Print record(s) from an input file.
- **Alter Data** Replace the data in any portion of the record with your supplied constant data, data from other locations in the record, or data from SELECTIT provided variables (date, jobname, datasetname, etc.).
- **Record reformat** A record can be completely reformatted, fields can be modified, deleted and/or added.

Reading Guide

The Basic User's Guide

This guide covers the basic functions of SELECTIT. It can be used as a self-contained document as it makes no reference to the other two sections. Optional features are not covered and not all commands or operands of commands are documented. The intent of the Basic User's Guide it to get a new SELECTIT user up and going as quickly as possible.

The Advanced User's Guide

This guide is for anyone who is comfortable with the basic SELECTIT functions and is now ready to explore and utilize some of the advanced features. The full descriptions of all syntax options and all commands and operands are covered.

Advanced Topics

This section provides additional background information to assist you in understanding how SELECTIT works and how your commands 'fit in' to the process.

The section (like the Basic User's Guide) contains numerous examples of SELECTIT runs to guide you in its use.

Basic User Guide

The Basic User's Guide contains the following sections:

SELECTIT Overview

Summary

The Basic User's Guide should provide you with the necessary information to perform the vast majority of your file manipulation needs with SELECTIT. You will find information on how to perform functions such as:

- A simple copy of one file to another.
- Printing the contents of a file.
- Selecting only certain records for printing or copying.
- Modifying data while copying a file, either globally or on a selective basis.
- Converting a file from one RECFM to another (e.g. fixed to variable, etc.).
- Splitting a file into 2 or more other files.
- Altering the layout of a file (i.e. moving fields around).

Sample runs showing the JCL and SELECTIT coding needed to perform the above examples can be found in "Basic Usage Examples".

Keep it Simple

SELECTIT has always attempted to remain 'easy to use'. Numerous versions and releases have however added much additional power and flexibility to the utility. Always keep in mind though that the simple functions are just as simple to invoke in the current version of SELECTIT as they were in the original release. JCL and control cards set up to run many years ago with the original release will still function perfectly with the current release.

If you are attempting a simple function and are getting bogged down in trying to determine how to do it, back up a bit and examine some of the examples which are provided. The solution is bound to be simpler than you think.

Items SELECTIT does to simplify things for you:

- Simple JCL. e.g. a simple file copy requires only 2 DD statements ("JCL Requirements" contains full details on JCL requirements).
- Automatic provision of DCB parameters for the output dataset. If you provide none in the JCL, SELECTIT will copy them from the input file. If you provide a partial set of parameters, SELECTIT will provide only the missing ones. Note the automatic provision of BLKSIZE may be suppressed if you wish it to be chosen by the operating system.
- Automatic handling of concatenation of unlike devices. As well, statistics are provided individually for each dataset in the concatenation.
- Automatic record format and dataset organization conversion. SELECTIT can copy from any normal record format (F/V/U) to any other record format and supported dataset organization. When variable format records are involved, the RDW (Record Descriptor Word) is either created or removed as appropriate.

SELECTIT's Command Language

The command language which is used in SELECTIT has certain anomalies from other 'programming' languages which are admittedly a little peculiar. Most of these were created as a by-product of the way SELECTIT evolved. New functions and facilities were always added in such a way that complete upward compatibility was maintained with prior releases.

For users who are familiar with other utilities (especially copy utilities), and with high-level Programming Languages (COBOL, for example), some of SELECTIT's language features may not be intuitively obvious.

- All statements end with a period ('.').
- All literal constants are enclosed within single quotes (apostrophes) and must be preceded by a format code (for example, you must use C'A', rather than just 'A').
- Labels on statements must be followed by a colon (':') and a blank.
- You reference fields by location and length. For example, positions 73-80 (field length 8) are referenced by 73/8. You may optionally prefix the field location with alphabetic comments SEQNUM73/8). Full details on coding field definitions is in "Field Definitions (Buffer operands)".
- You do not normally request the reading of records yourself with SELECTIT statements. Think of your control statements as executing within a SELECTIT controlled main line in which the record is read automatically from the primary input file (INPUT1), and control is passed to your control statements. The following is the processing automatically done within SELECTIT. i.e. there is no need to code commands to actually read your input file.

```
RETUR  Read next INPUT1 record
N:     At end of file, go to EOJ.
       Copy record to Output buffer.

>>>>>  Your control statements here <<<<<<<<
>>

       GOTO RETURN.

EOJ:   Perform termination processing.
```

The complete description of SELECTIT syntax is covered in "Basic Syntax Rules".

JCL Requirements

```
    //jobname   JOB (accting-data),SAMPLE.JCL
    //stepname EXEC PGM=SELECTIT
[   //SYSPRINT   DD SYSOUT=*             ]
    //INPUT1     DD DSN=...,DISP=...
[   //OUTPUT1    DD DSN=...,DISP=...     ]
[   //SYSIN      DD *                    ]
[   SELECTIT commands                     ]
[   /*                                    ]
```

Default Function Selection

If the SYSIN file is not present, is specified as DUMMY, or contains no records, SELECTIT will examine the remainder of the JCL present and attempt to automatically select a default function to perform.

In order to make this automatic selection, the following DD names are associated with the following purposes:

SYSPRINT Is used for all messages from SELECTIT along with any requested file dumps. It will also contain statistics related to the I/O performed against all DD's. SYSPRINT is optional. If absent, SELECTIT will simply not produce any messages or file statistics.

INPUT1 Is the primary input file to SELECTIT. It can be allocated to any sequential or keyed file.

Note: If allocated to a VSAM keyed file, the file will be treated as if it were a RECFM=V type dataset. i.e. all records will be prefixed with a 4 byte RDW (Record Descriptor Word)

OUTPUT1 Is the primary output file. For sequential or ISAM datasets SELECTIT will automatically complete any missing DCB information by copying it from the characteristics of the INPUT1 file. For keyed VSAM, the information must have been provided during the IDCAMS DEFINE of the dataset.

SYSIN Is used for your SELECTIT commands. If control statements are present here, all automatic selection of the function to be performed is suspended.

The two simplest functions of SELECTIT are the copying or printing of a dataset. When no control cards are present, SELECTIT will choose one of these functions based on the following:

- INPUT1 and OUTPUT1 present SELECTIT will copy all records from INPUT1 to OUTPUT1.
- INPUT1, OUTPUT1 and SYSPRINT present SELECTIT will copy all records from INPUT1 to OUTPUT1. Statistics of the number of records copied will be displayed on SYSPRINT.
- INPUT1 and SYSPRINT present SELECTIT will print all records from INPUT1 on SYSPRINT in full DUMP format. i.e. Hex and Character format. Statistics related to the operation will be displayed on SYSPRINT.

Any other combination SELECTIT will terminate with an error message on SYSPRINT (if present) or in the JOB log if SYSPRINT is not available.

Note: When performing a full copy of a file, allow SELECTIT to automatically select the copy function. Performance will be improved since SELECTIT 'knows' that no interaction with user control commands is required, and therefore will use an optimized function. Further discussion of performance items will be found in "Basic Performance Considerations".

Basic Syntax Rules

Although SELECTIT can perform the basic functions of Copying or Dumping a file without control cards (see "JCL Requirements"), anything more involved than that requires you to supply some control commands to direct SELECTIT's actions.

A SELECTIT command may optionally be prefixed by a label with which it can be referenced by other commands; and will typically be followed by one or more operands.

Format The general format of SELECTIT commands is as follows:

```
[label:]  command  [command-operands ... ] .
```

Command Syntax

- All commands are completely free format and use columns 1 to 72 of the control card. Columns 73 to 80 are ignored by SELECTIT.
- All commands follow a 'sentence' structure. i.e. commands and operands are separated by space(s) and the last 'word' of the sentence is terminated by a period. (COBOL users should feel quite comfortable).
- A keyword type parameter must not be separated from its value by any intervening space. e.g. COPYBUFF(YES) is valid but COPYBUFF (YES) is not.
- Sentences may begin and end without regard to statement boundaries. i.e. More than one sentence may be present on a single statement or a single sentence may span as many statements as desired.

Continuation Rules

- Continuations may be made at any point, either between 'words' or even in the middle of a 'word'. Common sense would be to avoid continuations in the middle of a 'word', the result usually is quite unreadable control commands. Since SELECTIT allows unlimited blanks between 'words' (even entire blank lines), continuations between words is much more natural.
- If you do wish to continue a command in the middle of a 'word' (e.g. when the 'word' is a very long literal), simply continue coding right through column 72 into column 1 of the next card. SELECTIT treats column 1 of a card as being logically adjacent to column 72 of the previous card.

  For example, if the word SELECT is to be continued across two cards starting from column 70. Columns 70-72 of the 1st card would contain 'SEL' and the characters 'ECT' would be in columns 1-3 of the next card.
- If you are continuing a quoted string, remember that columns 72 and column 1 of the next card are logically adjacent, the data in the columns may be blank or non-blank as required by the quoted string.

Labels

SELECTIT allows commands to be assigned labels (for use in GOTO commands). Any command may be preceded by a label of the format:

```
xxxxxxxx:
```

The label may be of any length, although only the 1st 8 characters are significant to SELECTIT. It must be terminated with a colon **and** a space and may consist of any alphanumeric characters except the following: quotes ('), slashes (/), colons (:), parentheses (), or blanks.

A command may have more than one label, for example:

```
START:
BEGIN: command ...
```

Keywords used as labels.

Although SELECTIT has no specific restrictions on using any of its keywords as labels, we recommend you avoid them. They would merely introduce potential confusion. Two keywords are predefined however and must be avoided, they are **RETURN**: and **EOJ**. See "GOTO".

Comments

Comments may be entered in a SELECTIT program wherever appropriate to further document the SELECTIT control commands. A comment also follows SELECTIT sentence format rules; it must start with the reserved label ***:** and be terminated by a period following the last word of the comment. The ***:** must be followed by at least one space before the comment itself.

```
I RTYPE3 = C'A'          *: THIS IS A COMMENT.
F
  SELECT TODD(OUTPUT1).  *: ANOTHER COMMENT.
```

Note: Two common errors are made in entering SELECTIT comments; the first is the inadvertent use of single quotes, the second is the accidental use of a period before the end of the comment.

For example do not use words like "it's", "Fred's", etc. SELECTIT will take the single quote to be the start of a quoted string and scan ahead looking for the closing quote. This could, for example, cause SELECTIT to scan past the period which was intended to end the comment sentence.

Literals

Various commands require you to code literals as one or more of the operands. The format of literals is the same for all of the commands, and is designed to follow normal Assembler coding conventions (with some minor restrictions).

Format The general format of SELECTIT literals is:

```
    [r]f[Lnnn]'value'
```

where:

r is the repeat factor, which must be numeric; if omitted, the default is 1. See also the restriction under **Lnnn** below.

f is the format, and must be one of the following entered as an upper case alphabetic:

- **C** Character
- **F** Binary Fullword
- **H** Binary Halfword
- **P** Packed Decimal
- **X** Hexadecimal

Lnnn may optionally specify the length of the literal for types C, P, and X. If not coded, the effective length of the value within the apostrophes is used.

The **'L'** must be entered in upper case.
The maximum length for types C and X is 256 bytes; the maximum length for type P is 16 bytes.
You must omit the the length factor for types F and H.
If the repeat factor is used, it must not be greater than 256, AND the product (**r*nnn**) also must not be greater than 256.
'value' contains the actual literal constant; it must be coded within apostrophes.

Valid characters in the 'value' for the literal types are:

'C' - any EBCDIC characters. If an apostrophe forms part of the value, it must be coded as two consecutive apostrophes.

'F, H, P' - Any numerics (0-9). If a negative number is required, the value must be preceded by a '-'; positive values must not be preceded by a '+'.

'X' - Any hexadecimal digits (0-9, A-F). You must specify an even number of hex digits in the 'value' literal. Hex digits A-F must be entered in upper case.

Examples

Coded Interpreted Value

Coded as	Interpreted as
C'ABC'	C'ABC'
C'abc'	C'abc'
5C'A'	C'AAAAA'
CL10'XXX'	C'XXX ' (7 trailing spaces)
F'80'	X'00000050'
H'-4'	X'FFFC'
P'12'	X'012C' ('C' sign - see note)
X'012F'	X'012F' ('F' sign - see note)
PL3'-5'	X'00005D'
2PL2'1'	X'001C001C'
XL4'0A98'	X'00000A98'

Note: 'F' and 'C' signs will compare equal in a packed comparison (IF), but you should use an 'X' format literal for ALTER commands which require an 'F' sign in the output field.

A packed comparison will be performed only if one of the operands is a P-type literal. In all other cases, a character comparison will be performed.

Error Examples

Coded	Reason for Error
CL1000'X'	length factor exceeds 256
cL20'a'	'c' not upper case; 'a' is valid
Cl20'a'	'l' not upper case; 'a' is valid
400C'E'	repeat factor exceeds 256
200CL16'WW'	repeat*length exceeds 256
HL1'5'	length factor not allowed
PL17'2'	length factor exceeds 16
200PL5'20'	repeat*length exceeds 256
P'5C'	'C' is not a valid numeric
XL1000'CA'	length factor exceeds 256
X'12345'	odd number of digits in value
X'BCDEFG'	'G' is not a valid hex digit
X'89ab'	digits 'a' and 'b' not upper case

Field Definitions (Buffer Operands)

To perform any functions more complicated than simple Copy and/or File Dump operations, you will probably need to create control cards which direct SELECTIT operations based on the data contents of the records being processed.

All SELECTIT commands which reference data in the records being processed use the same syntax to refer to that data.

Field Location

All fields are referenced based on their location within the record. When referencing a field in the buffer, the operand is coded as a non-zero numeric value representing the starting location (i.e. leftmost byte) of the field within the record.

The 1st data byte of the record is 1, not 0. Keep in mind when processing RECFM=V type records that SELECTIT includes the RDW (Record Descriptor Word) in the record so that a value of 1 refers to the RDW itself; the 1st real data byte of the record is at location 5.

You may optionally prefix the numeric value with any number of alphabetic characters to serve as a comment. For example, an account number starting in location 43 of the record can be referred to as:

```
43 or
ACCT43 or
ACCOUNTNUMBER43
```

The prefix may not contain any embedded numeric values. For example a 30 days arrears value in location 25 can not be referred to as

```
ARREARS30DAYS25
```

but rather as

```
ARREARSTHIRTY25
```

Field Lengths

Depending on the primary SELECTIT command used, and the format of other operands of the command, you may be required to specify a length value for the field.

Example: If you are comparing a field in a record to a literal, a length would not be needed, SELECTIT will use the effective length of the literal to which the field is being compared. If however you are comparing or moving fields within the record, SELECTIT has no way of determining the length of the fields. It is up to you to provide this information.

The length of a field is provided by adding a suffix to the location operand. This suffix consists of a slash (/) followed by the desired length in bytes. Using the same fields as in the examples above, lets assume the account number is 6 bytes long and the arrears field is 7 bytes long. For operands which require a length value to be supplied, the operands would appear as:

```
43/6 or
ACCT43/6 or
ACCOUNTNUMBER43/6

25/7 or
ARREARSTHIRTY25/7
```

The Basic Commands

The Commands described here are only a sub-set of the full SELECTIT command set. The data provided does not cover all variations and options of the commands but will provide the information you require to perform the vast majority of your requirements.

Note that if SYSIN is present at least one SELECT command is required, all other commands are optional.

These four commands, although basic, still provide you with a lot of power. They are:

Basic IF command

Basic ALTER command

Basic GOTO command

Basic SELECT command

IF Command (Basic)

Format

```
[label:]  IF  operand-a  test  operand-b
    {AND¦OR}  operand-a  test operand-b] ... ]
          [THEN] {ALTER command. |
                  GOTO label.     |
                  SELECT command.}
```

Function

To test the contents of field(s) of the input data records for specified values, and conditionally perform other commands.

Operands

operand-a operand-b	Each of these may represent: ○ a field within the input record; or ○ a literal value. Refer to "Field Definitions (Buffer operands)" and "Literals" for coding format. A length value may be present on either operand. The length value must be omitted for literals. **Note**: Either operand-a or operand-b may be a literal or a buffer operand. The only combination which is illogical is the case where both operands are literals.

test	Specifies the test to be made. It may be either an abbreviation or symbol as follows:
	`EQ  =  Equal to`
	`GE  >=  Greater than or equal to`
	`GT  >  Greater than`
	`LE  <=  Less than or equal to`
	`LT  <  Less than`
	`NE  ¬= <>  Not equal to`
	`NGE  ¬>=  Not greater than or equal to (same as LT)`
	`NGT  ¬>  Not greater than (same as LE)`
	`NLE  ¬<=  Not less than or equal to (same as GT)`
	`NLT  ¬<  Not less than (same as GE)`

	AND/OR Multiple tests may be made in any AND/OR relationship. The AND/OR relationships are treated in the same manner as COBOL 'IF' statements (i.e. AND takes precedence over OR, otherwise left to right). Parentheses may **not** be coded to override this hierarchy.
	
	Implied location/operator
	If multiple tests are being made against a field, the first operand may be omitted if it is the same as the one in the preceding test; both the first operand and the test may be omitted, if both are identical to those in the previous test.
	
	The following are all valid ways of specifying the same test:
	
	`IF P10 > C'A' AND P10 < C'Z' OR P10 = C'.'`
	`IF P10 > C'A' AND < C'Z' OR P10 = C'.'`
	`IF P10 > C'A' AND < C'Z' OR = C'.'`
	
	Similarly:
	
	`IF P10 = C'A' OR P10 = C'B' OR P10 = C'C'`
	`IF P10 = C'A' OR C'B' OR C'C'`
	
	And:
	
	`IF P10 ¬= C'A' AND P10 ¬= C'B' AND P10 ¬= C'C'`
	`IF P10 ¬= C'A' AND C'B' AND C'C'`
	
	Note the use of AND in conjunction with '¬='; this is the inverse of the previous example, which used OR and '='.

[THEN] command	The command to be executed (i.e. ALTER, GOTO, or SELECT) if a true result occurs, is coded exactly the same as the unconditional form of the command. Note that an 'IF' may not be coded directly after THEN.
	
	To improve readability of the IF command, the word THEN may be placed before the action to be taken, but is not required; for example, the two commands below are identical in function:
	
	`IF POS20 = C'X' THEN GOTO ...`
	`IF POS20 = C'X' GOTO ...`

Data Types

All comparisons performed by 'IF' are logical compares with the exception of compares on Packed Decimal fields; these are performed with a Compare Packed (CP) instruction to obtain a proper algebraic compare. SELECTIT determines whether to use a Packed Compare or not by the type of literal used in the 'IF' command. Use a 'P' type literal when you wish a Packed Decimal compare.

Example: If you wish to compare a 2 byte packed field for zero, use

```
IF FIELD10 = PL2'0' THEN...
```

do not use

```
IF FIELD10 = X'000C' THEN...
```

since this would cause a logical compare to be used and if, for example, the field contained X'000F' (a valid packed decimal zero) the 'IF' would be FALSE.

Examples

```
IF ACCTBAL33 > PL4'100' THEN GOTO RETURN.

IF COLUMN20 = C'XX' ALTER COL20 COPY C'44'.

IF STATUS10 = C'A' OR C'B' OR C'C' GOTO RETURN.

IF 20 GT P'50' AND 30 NE C'X'
OR 50 EQ X'F3CE' THEN SELECT TODD(OUTPUT2).

IF BALANCE1 > F'1000' AND STATUS10 = C'A' GOTO COPYIT.

IF COL20 = COL40/5 GOTO EQUAL.
```

ALTER Command (Basic)

The ALTER command is used to modify the data in a record. It allows you to replace portions of the data with constant information (i.e. literals) or with information from other locations in the record.

Format

```
[label:]  ALTER  target-operand  action  source-operand.
```

Operands

target-operand	This represents the leftmost byte of the field within the record which is to be altered. See "Field Definitions (Buffer operands)" for a description of coding this field.
action	**COPY** Replace the target-operand with the source-operand. **REPLACE** (alias for **COPY**)
source-operand	This may represent a field within the record or a literal value. A length value (the ../nn portion of an operand) must be coded for buffer field references; it must be omitted for literals (literal lengths are implicit in their definition). See "Literals" for instructions on coding literals; see "Field Definitions (Buffer operands)" for instructions on coding field references and length values.

Examples

```
ALTER BALANCE30 COPY PL5'0'.

ALTER COL10 COPY COL20/10.

ALTER POSITION2 COPY POSITION3/20.

ALTER COL10/10 COPY COL20.
```

GOTO Command (Basic)

The GOTO command is used to transfer control to another command in the SELECTIT command stream.

Format

```
[label:]  GOTO  label.
```

Function

Transfers control to the specified label, which may be a user-defined label, or one of the special labels RETURN and EOJ. Refer to "Basic Syntax Rules" for valid label syntax.

Special Labels

Two special reserved labels are automatically created by SELECTIT.

RETURN	A transfer to this label will terminate processing of the current input record. If the record has not already been SELECTed by some other control statement, it will not be processed any further. Processing will continue at the start of your SELECTIT commands with the next input record from INPUT1.
EOJ	A transfer to this label will immediately terminate the program even if all requested functions are not complete. Refer to the topic 'Normal Termination Rules' for more information as to how SELECTIT determines when to terminate.

Note that an error condition will be raised if you explicitly code labels with the name RETURN or EOJ in your SYSIN stream.

In order to avoid possible confusion, we recommend that you do not use any of SELECTIT's reserved words as labels.

SELECT Command (Basic)

Format

```
[label:]  SELECT  [RECS({ALL|nnn})]
                  [SREC(sss)]
                  [TODD(ddname)]
                  [PRINT({CHAR|DUMP|HEX})]
                  [GOTO(label)]
```

Function

The SELECT command controls the output processing to be performed on selected records. Records may be written to an output file (via the TODD operand) or may be printed on the SYSPRINT file in one of three formats (the PRINT operand). When used with the TODD operand, think of it functionally as a WRITE command.

You may optionally request that only a certain number of records be processed (the RECS operand) or that processing is to begin at other than the first record (the SREC operand).

The optional GOTO operand allows you to specify a SELECTIT label to which control should be given as an alternative to the next sequential SELECTIT command.

Coding Restrictions

Although TODD, PRINT, and GOTO are all shown as optional keywords, at least one of them must be present.

If SYSIN is present, at least one SELECT command with either the TODD or PRINT operand (or both) is required.

DD Statement Requirements

If the TODD operand is coded, there must be a DD statement present for the specified ddname.
If the PRINT operand is coded, the SYSPRINT DD statement must be present in the JCL.

Operands

RECS(ALL) | RECS(nnn)	Any numeric value or 'ALL' (The default is 'ALL'). Specifies the number of records (nnn) this SELECT is to process. If 'nnn' is greater than the number of records in the input file, it will be treated as though 'ALL' had been specified. **Note**: The RECS(nnn) count is matched against the number of times that this SELECT command has been **executed**, not the number of records read from Input files. e.g. If the SELECT is the condi tional command of an 'IF' statement, only the records which have already satisfied the 'IF' are counted.

SREC(sss)	Specifies the record count (sss) of the primary input (INPUT1) at which this SELECT will start processing records. If the SREC operand is omitted, SREC(1) will be assumed.
TODD(ddname)	Requests that the record currently in the Output buffer be written to the named output file ('ddname'). SELECTIT automatically copies all input records to the output buffer prior to passing control to your commands.
GOTO(label) | GOTO label	Following any other requested actions (i.e. TODD and/or PRINT) for the SELECTed records, control is passed to the named label.

Action Commands

A SELECT command may contain any combination of TODD, PRINT, and GOTO action operands, but not more than one of each. It must contain at least one of them.

The PRINT, TODD, and GOTO commands may be coded in any order, but PRINT and TODD will be processed prior to the GOTO (if one is present).

Normal Termination Rules

SELECTIT 'knows' what SELECT commands you have entered and uses this knowledge to terminate as soon as all requested output has been produced.

Example: If any of your SELECT commands specify, or have defaulted to, RECS(ALL), then SELECTIT will not terminate until end-of-file on the primary input file (INPUT1). If all coded SELECT commands have some limit specified by RECS(...), then when all the SELECT commands have reached their limiting RECS(...) value, the step will terminate, even if all records from the input file have not been processed.

This logic is designed to avoid wasted reading of the remainder of a file once all desired output has been created.

Performance Considerations

For most uses of SELECTIT, you should only concern yourself with optimization when you will be processing very large datasets. When this is the case, the next two sections should be reviewed.

- Take the Defaults
- JCL BUFNO

Take The Defaults

When you are using SELECTIT to copy an entire file, and no selection criteria or record modifications are made, **always** let SELECTIT choose the automatic default of a full copy. See "Default Function Selection".

Take the following 2 examples:
Example 1.

```
//STEP1    EXEC PGM=SELECTIT
//SYSPRINT   DD SYSOUT=*
//INPUT1     DD DSN=INPUT.DATASET,DISP=SHR
//OUTPUT1    DD DSN=OUTPUT.DATASET,DISP=(,KEEP),
//               UNIT=..., etc.
//SYSIN      DD *
SELECT RECS(ALL) TODD(OUTPUT1).
/*
```

Example 2.

```
//STEP1    EXEC PGM=SELECTIT
//SYSPRINT   DD SYSOUT=*
//INPUT1     DD DSN=INPUT.DATASET,DISP=SHR
//OUTPUT1    DD DSN=OUTPUT.DATASET,DISP=(,KEEP),
//               UNIT=..., etc.
```

The two examples appear to be functionally similar; after all, SELECTIT will default to use

```
SELECT RECS(ALL) TODD(OUTPUT1).
```

in Example 2 since both INPUT1 and OUTPUT1 DD's are present.

True, both examples will copy the file correctly. The second example will however perform it more efficiently. Since SELECTIT 'knows' it is creating its own internal default, it is able to avoid generating the small additional overhead code needed to interface to the commands which a user supplies on SYSIN. Although a small overhead, it can add up when copying files with many thousands of records since it is incurred for every record.

Moral: Take the default.

JCL BUFNO

When the files you are processing with SELECTIT are either very large or are poorly blocked, increasing the BUFNO parameter for the file in the JCL can have dramatic effects on the elapsed time needed to read or write the file. Increasing the BUFNO will of course require you to review the REGION parameter to ensure that you do not cause an S80A abend (insufficient region).

A general guideline for poorly blocked disk datasets is to increase the BUFNO to one that will cause processing to take place approximately 1 track at a time.
Example: For a blocksize of 200 bytes on a 3380, use a BUFNO of around 70. A blocksize of 3120 would need a BUFNO of about 13.

Tape datasets should be set so that (blksize * BUFNO) is approximately 70K - 90K.

These are only very rough guidelines. In some cases experimentation may be needed to determine what values give you the optimum price/performance ratio.

Information on all other JCL parameters is in "JCL Requirements".

Basic Usage Examples

The following pages contain examples of basic SELECTIT usage, most of your normal requirements will be covered by one of these examples.

Copy an Entire Dataset

In this example the entire file INPUT.DATASET is copied to another file OUTPUT.DATASET.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=INPUT.DATASET,DISP=SHR
//OUTPUT1   DD DSN=OUTPUT.DATASET,DISP=(,KEEP),UNIT=..., etc.
```

Notes:

1. The SYSPRINT DD is optional, since it is present, SELECTIT will use it to display statistics related to the copy (records read, records written, DCB characteristics, etc).

2. The INPUT1 DD as shown is allocated to a single dataset. SELECTIT will handle concatenations of datasets on different devices and with different DCB characteristics, in any order. If statistics are produced, SELECTIT will report on each dataset in the concatenation separately.

3. The input dataset may be any sequential, ISAM or keyed VSAM dataset.

4. The output dataset may be any sequential, ISAM or keyed VSAM dataset. Note that the ISAM and keyed VSAM datasets are treated as being initially loaded. i.e. you can not use SELECTIT to add records to an existing file.

5. Any combination of input and output dataset organizations is supported. You may copy from sequential to sequential, sequential to ISAM, VSAM to sequential, ISAM to ISAM, ISAM to sequential, etc. and is determined automatically by your JCL (no special commands are required).

Print (Dump) an Entire Dataset

In this example the entire file TEST.DATAFILE will be printed on SYSPRINT in both character and hexadecimal format.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=TEST.DATAFILE,DISP=SHR
```

Notes:

1. The DUMP option will be selected automatically since a) No SYSIN DD was provided and b) only SYSPRINT and INPUT1 DD's are present. Therefore the only logical function possible is a file dump.

2. If the data content of the file is such that a different print format from the default (DUMP) is more suitable, simply add a SYSIN DD as follows:

   ```
   //SYSIN DD *
   SELECT PRINT(format).
   /*
   ```

   where 'format' is CHAR for character only format, or HEX for hexadecimal only format.

Selective Copy and/or Dump of a Dataset

In this example records from the file PROD.MASTER will be selected based on the record type field which is in byte 3 of the record. Record types 1 and 3 will be copied to an output file; record type 2 will be dumped to SYSPRINT in character format.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=PROD.MASTER,DISP=SHR
//OUTPUT1   DD DSN=XTRACT.FILE,DISP=(,KEEP),UNIT=..., etc.
//SYSIN     DD *
IF RECTYPE3 = C'1' OR C'3'
   THEN SELECT TODD(OUTPUT1).
IF RECTYPE3 = C'2'
   THEN SELECT PRINT(CHAR).
/*
```

Notes:

1. The first IF command takes advantage of the implied operand facility. It could also have been coded in either of the following manners:

   ```
   IF RECTYPE3 = C'1' OR = C'3'
    or
   IF RECTYPE3 = C'1' OR RECTYPE3 = C'3'
   ```

2. The THEN keyword, which serves only to improve readability of the commands, could be omitted.

   i.e.
   ```
   IF RECTYPE3 = C'1' OR C'3'
         SELECT TODD(OUTPUT1).
   ```

   is an equally valid statement.

Deleting specified records

In this example obsolete records will be deleted during the copy of the TRAN.HISTORY file. Obsolete records are those with a BALANCE field of zero and an INACTIVE period greater then 90 days. The BALANCE field is in location 134 of the record; the INACTIVE field is in location 22. Both are 4 byte packed decimal fields.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=TRAN.HISTORY(0),DISP=SHR
//OUTPUT1   DD DSN=TRAN.HISTORY(+1),DISP=(,CATLG),UNIT=..., etc.
//SYSIN     DD *
IF BALANCE134 = PL4'0' AND INACTIVE22 > PL4'90'
   THEN GOTO RETURN.
SELECT TODD(OUTPUT1).
/*
```

Notes:

1. The deletion of records is accomplished by simply **not** SELECTing the unwanted records during a copy operation. In this example, the unwanted records are found by the compound IF command and control is passed by the GOTO command to the SELECTIT reserved label 'RETURN'. Passing control to the label RETURN indicates to SELECTIT that all processing of the current record is complete.

2. Since only records which fail the IF command will reach the SELECT command, the obsolete records are dropped from the output dataset.

Modifying all records in a file

In this example year-to-date fields in a master must be reset to zero at year end. Two fields must be reset: TRANSACT, a 4 byte binary field at location 12; and TOTPURCH, a 5 bytes packed decimal field at location 37.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=YREND.MASTER,DISP=SHR
//OUTPUT1   DD DSN=NEWYR.MASTER,DISP=(,CATLG),UNIT=..., etc.
//SYSIN     DD *
ALTER TRANSACT12 COPY F'0'.
ALTER TOTPURCH37 COPY PL5'0'.
SELECT TODD(OUTPUT1).
/*
```

Notes:

1. Since there are no conditional commands (i.e. IF's), the two ALTER commands will be executed for every record in the file before the record is written to the output file.

Modifying selected records in a file

In this example company divisions have been consolidated. All records related to the former ONTARIO and QUEBEC divisions must be modified to the new EASTERN division name. Records for the maritime provinces are all to be in the new MARITIME division. Records for all other divisions are not to be modified. The division name field is 20 bytes long starting at location 33.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=OLD.DIVNFILE,DISP=SHR
//OUTPUT1   DD DSN=NEW.DIVNFILE,DISP=(,CATLG),UNIT=..., etc.
//SYSIN     DD *
IF DIVISION33 = C'ONTARIO' OR C'QUEBEC'
   THEN ALTER DIVISION33 COPY CL20'EASTERN'.
IF DIVISION33 = C'NOVA SCOTIA' OR
                C'NEW BRUNSWICK' OR
                C'P. E. I.' OR
                C'NEWFOUNDLAND'
   THEN ALTER DIVISION33 COPY CL20'MARITIME'.
SELECT TODD(OUTPUT1).
/*
```

Notes:

1. Records for all other divisions will fail both IF commands and will be written unchanged to the OUTPUT1 file.

Splitting a file

In this example an existing file must be split in two. All record type 0 records will be written to both output files. Record types 1, and 2 are to be directed to separate output files. The record type is a binary halfword in position 1 of the record. Any other record types are to be ignored.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=COMBINED.FILE,DISP=SHR
//RTYPE1    DD DSN=RECTYP1.FILE,DISP=(,CATLG),UNIT=..., etc.
//RTYPE2    DD DSN=RECTYP2.FILE,DISP=(,CATLG),UNIT=..., etc.
//SYSIN     DD *
IF RECTYPE1 = H'1' THEN SELECT TODD(RTYPE1) GOTO RETURN.
IF RECTYPE1 = H'2' THEN SELECT TODD(RTYPE2) GOTO RETURN.
IF RECTYPE1 ¬= H'0' THEN GOTO RETURN.
SELECT TODD(RTYPE1).
SELECT TODD(RTYPE2).
/*
```

Notes:

1. The output DD names chosen were RTYPE1, and RTYPE2. There is no requirement to use OUTPUT1, OUTPUT2, etc. format names.

2. The logic for record type 0 demonstrates how multiple SELECT commands can be used to repeatedly process the same record. A SELECT TODD(...) does not alter or clear the data in any way. In fact you may:
   - modify the record
   - write it to a file
   - further modify the record
   - write again to the same (or another) file
   - and perform these functions repeatedly in any order or combinations you desire.

Altering the RECFM of a Dataset

In this example a full copy of VARIABLE.DATASET is made to FIXED.DATASET. The record format of the file is altered during the copy from RECFM=V to RECFM=F.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=VARIABLE.DATASET,DISP=SHR
//OUTPUT1   DD
DSN=FIXED.DATASET,DISP=(,KEEP),DCB=(RECFM=FB,LRECL=nnn,BLKSIZE=yyy),
//             UNIT=..., etc.
```

Notes:

1. SELECTIT will automatically remove the RDW (Record Descriptor Word) from the input data and write only the data portion of the record to the output file.

2. If the input record length is less than the output LRECL, the record will be padded with blanks. If the record is longer, it will be truncated. No warning is issued if truncation occurs.

3. SELECTIT will correctly handle conversions between any standard RECFM's. RDW's will be either dropped or created as necessary to accomplish the conversion.

Altering the layout of a record

In this example a file is to be converted from

Old Layout		New Layout	
PARTNO	4 bytes	QTY	4 bytes
QTY	4 bytes	PARTNO	5 bytes
STAT	1 byte	DESCR	10 bytes
DESCR	10 bytes		

The STAT field is to be eliminated and the spare byte added to the PARTNO field as a leading zero. The QTY field is to be moved to the beginning of the record

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=OLDFMT.FILE,DISP=SHR
//OUTPUT1   DD DSN=NEWFMT.FILE,DISP=(,CATLG),UNIT=..., etc.
//SYSIN     DD *
ALTER NEWQTY1 COPY OLDQTY5/4.
ALTER NEWPFILL5 COPY C'0'.
ALTER NEWPARTNO6 COPY OLDPART1/4.
SELECT TODD(OUTPUT1).
/*
```

Notes:

1. The ALTER commands which move the fields all have the length operand coded since the data being moved is not a literal.

2. The order of ALTER commands does not matter. Even though the first ALTER would seem to have overlaid the old PARTNO field with the new QTY field (destroying PARTNO) this is not the case. SELECTIT has full access throughout the process to the original input record. You need not concern yourself with overlapping fields such as these.

Creating a test file

In this example a test data file is to be extracted from a production file. The production file contains a variety of record types and the desired test file is to contain 50 each of the record types 1, 3, 5, and 7. All other record types are to be ignored.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=PRODUCT.DATA,DISP=SHR
//TDATA     DD DSN=TEST.DATA,DISP=(,KEEP),UNIT=..., etc.
//SYSIN     DD *
IF RTYPE1 = H'1' SELECT RECS(50) TODD(TDATA) GOTO RETURN.
IF RTYPE1 = H'3' SELECT RECS(50) TODD(TDATA) GOTO RETURN.
IF RTYPE1 = H'5' SELECT RECS(50) TODD(TDATA) GOTO RETURN.
IF RTYPE1 = H'7' SELECT RECS(50) TODD(TDATA) GOTO RETURN.
/*
```

Notes:

1. Each SELECT command includes the RECS(...) operand to override the normal default of ALL. Once a SELECT command has processed the specified number of records, it effectively 'disappears'.

2. Each SELECT command also uses the optional GOTO operand to transfer control to RETURN to avoid 'falling through' to alternative IF commands which can never be satisfied.

3. SELECTIT will automatically detect when all specified SELECT's have completed their processing and will terminate immediately. This avoids the reading of the remainder of the INPUT1 file when it is not possible for any of the records to be selected to an output file.

Advanced User Guide

The Advanced User's Guide provides the information you need to utilize all the power and flexibility of SELECTIT. You will find there are additional primary commands and many optional operands for some of the commands you became familiar with in the Basic User's Guide.

Some of the advanced features described are:

- The use of IF ... THEN ... ELSE ... support.
- The use of DO ... END.
- The use of GOSUB ... GOBACK.
- How to scan for and manipulate data in a record without fixed layout positions.
- The handling of multiple input and output files.
- How to process all members of a PDS.
- How to control the OPENing and CLOSEing of input/output files.
- The use of User buffers and User Counters.
- The use of special 'pre-defined' fields and variables.
- The use of EQUATE symbols to simplify SELECTIT operand coding.
- Customizing SELECTIT operation and defaults (the OPTIONS) command.

JCL Requirements

```
    //jobname   JOB (accting-data),SAMPLE.JCL
    //stepname EXEC PGM=SELECTIT[,PARM='options']
  [ //SYSPRINT   DD SYSOUT=*          ]
    //INPUT1     DD DSN=...,DISP=...
  [ //INPUT2     DD DSN=...,DISP=... ]
    .
    .
  [ //INPUTn     DD DSN=...,DISP=... ]
  [ //OUTPUT1    DD DSN=...,DISP=... ]
  [ //anyname1   DD DSN=...,DISP=... ]
    .
    .
  [ //anynamen   DD DSN=...,DISP=... ]
    //SYSIN      DD *
    SELECTIT commands
    /*
```

EXEC statement The program name to be executed is SELECTIT. Optionally, a PARM value may be coded to specify various customization options for this run. This includes the passing of limited (8 bytes) of user data from the PARM field to the SELECTIT program. Further information on these options can be found in "Customizing the Run Time Environment" and "User PARM Data".

SYSPRINT The SYSPRINT DD statement is used by SELECTIT for all its hardcopy output. It may be allocated to any sequential output file and will support any standard record format (F, V, or U). If no DCB parameters are specified SELECTIT will default to using RECFM=FBA, LRECL=133, and a BLKSIZE of 5985 for tape or DASD allocations, or a BLKSIZE of 133 for SYSOUT allocations.

Unless you specifically request print output via SELECT PRINT(...) or DEBUG commands, the SYSPRINT DD is entirely optional.

INPUT1 INPUT1 describes the primary input file. It can be allocated to any sequential, ISAM, or keyed VSAM dataset.

INPUT1 may also be allocated to a PDS (not just a member of a PDS) when processing entire PDSs. See "Processing PDS Datasets" for special considerations when processing PDSs.

INPUT2-9 Secondary Input Files DDnames INPUT2 through INPUT9 are all optional, and describe secondary input files.

There is no requirement for consecutive numbering. For example, INPUT3 and INPUT7 could be present in the JCL, without INPUT2 or INPUT4/5/6.

If ISAM or VSAM keyed files are processed, the SKEY(..) and EKEY(..) functions of the

SELECT command do not apply, those functions apply only to the primary input file (INPUT1).

OUTPUT1 Default Output File DDname OUTPUT1 is the default output DDname, when SYSIN is absent and a copy operation is selected as the default operation. It may of course also be referenced explicitly by commands in the SYSIN stream.

Output Files (General) Output files (including OUTPUT1) may be present with any DDname of your choice.

Output Files (PDS Support) Output files (including OUTPUT1) may be allocated to a PDS (not just a member of a PDS) when processing entire PDSs. See "Processing PDS Datasets" for special considerations when processing PDSs.

Output Files (BLKSIZE) If you wish to suppress SELECTIT's copying of BLKSIZE from INPUT1 to the output files, see the CBLKSIZE operand in "OPTIONS" for more details.

SYSIN SYSIN may be allocated to any sequential, card image dataset (including a member of a PDS). It must contain your SELECTIT control commands.

Miscellaneous JCL Notes

The following items should be reviewed for certain special considerations. They are in no particular order.

- The output on SYSPRINT will consist of:
  - a listing of the SELECTIT control commands present on SYSIN
  - any diagnostics related to errors in the control commands
  - statistics related to the datasets processed during the run
  - listing of all non-zero UCTRs at termination. UCTRs are described in "Miscellaneous Fields".
  - all output specifically requested by SELECT PRINT(...) or DEBUG commands in the SYSIN input.

- When SYSPRINT is present, the OPTIONS parameters LIST(OFF) and STATS(OFF) can be used to selectively suppress some of the output. Details on this are in "Customizing the Run Time Environment".

- In most cases, INPUT1 is a required file and must contain at least one record. If INPUT1 is omitted, or consists of an empty file, then the following special considerations apply:
  - INPUT1 omitted: It is your responsibility to provide whatever control commands are needed in the SYSIN file to perform your desired actions.
  - INPUT1 empty or DD DUMMY or DSN=NULLFILE: Since the empty condition will cause an immediate end-of-file condition, SELECTIT would normally terminate before processing any of your input commands. To cause your control commands to be executed you must have an OPTIONS EOFLABEL(label) command coded, where 'label' is a user-defined label in the SYSIN stream. The option may be specified either in the SYSIN stream itself or as a PARM field value. See "Customizing the Run Time Environment" for further details.

- No input operations will be performed against a file referenced by DDnames INPUT2-9 in the JCL, unless

there is a GETNEXT for that DDname in the SYSIN stream. In particular, no OPEN will be issued for unreferenced INPUTn DDnames, although JCL disposition processing will be honored.

- In the event of a compile-time error, all referenced DDnames will normally have both OPEN and CLOSE issued against them. The exception to this will be DD's for which an OPTIONS DEFER(ddname) was specified.

- No operations will be performed against output DDnames in the JCL, unless there is a 'SELECT TODD(ddname)' for that DDname in the SYSIN stream. In particular, no OPEN will be issued for unreferenced DDnames, although JCL disposition processing will be honored; if an unreferenced DDname causes a new file to be created with a disposition of (,KEEP) or (,CATLG), unpredictable results will occur if a subsequent step tries to read it.

- As with Input files, in the event of a compile-time error, all referenced DDnames will normally have both OPEN and CLOSE issued against them. The same exception regarding OPTIONS DEFER(ddname) applies.

Advanced Syntax Rules

This section describes the full syntax of SELECTIT operands. If you are not a current SELECTIT user, or have not reviewed "Basic Syntax Rules", please do so now.

The majority of the differences between Basic SELECTIT syntax and Advanced syntax are apparent in the flexibility of the command operands, not in the format of the commands themselves.

Using the advanced syntax you will be able to:

- Access data in any of the buffers (Input or Output)
- Utilize user buffers for record storage or work areas
- Access SELECTIT created variables (Jobname, dataset names, date, time, volsers, etc.)
- Use user counters as dynamic variables to specify location and length of operands within buffers
- Use EQUATE to simplify operand coding.
- Request functions which treat data as free-form strings (i.e. search/modify data without 'knowing' its position within the record).

Quick Buffer Description

The syntax topics which follow are all related to how you may access data within the SELECTIT buffers. A quick summary of buffer usage is given here first to assist in understanding the syntax topics. A full description of buffer usage is provided by "Buffers".

Each input ddname has its own unique buffer so that data from all available input files can be accessed simultaneously. All output ddnames share one common output buffer. The input buffers are referenced by the ddname to which they are associated; the common output buffer is referred to via OBUFF. In addition, you may utilize up to 26 'user buffers' for any purpose you desire; these are referred to via UBUFFA through UBUFFZ.

Operand formats within SELECTIT can be broken down into three major categories:

'fixed' buffer operands This type of operand was described in "Field Definitions (Buffer operands)" . 'fixed' buffer operands are described by their location and length within a buffer.

'floating' buffer operands This type of operand allows you to 'scan' buffers for the presence (or absence) of some data value and optionally modify fields accordingly.

keyword variable operands This type of operand references a data field which has been pre-defined by SELECTIT and assigned a unique keyword 'name'. See "Pre-defined Keyword Variables" .

Fixed Buffer Operands

Fixed buffer operands are identical to the buffer operands as described in the Basic User's Guide (in which they were the only type of operand). This section will describe some additional techniques for coding these operands which allows you considerable extra flexibility.

The complete syntax of SELECTIT fixed buffer fields is as follows:

```
[prefix:][comment]{location | UCTRx}[/{length | UCTRy}]
```

prefix	The optional 'prefix' portion of the operand allows you to specify explicitly the buffer to which this operand refers. If not used, SELECTIT normally directs the operand to a default buffer as determined by the context of the operand as follows: **IF command operands** The operands are assumed to refer to the INPUT1 buffer. **ALTER destination operand** The operand is assumed to refer to the output buffer (OBUFF). **ALTER source operand** The operand is assumed to refer to the INPUT1 buffer. **'prefix'** may be one of the Input Buffers (INPUT1: through INPUT9:), the Output Buffer (OBUFF:), or one of the User Buffers (UBUFFA: through UBUFFZ:). The colon (**':'**) is required, and must be neither preceded nor followed by blanks.

comment	You may optionally prefix the numeric location value with any number of alphabetic characters to serve as a comment. For example, an account number starting in location 43 of the record can be referred to as:
	
	`43 or`
	`ACCT43 or`
	`ACCOUNTNUMBER43`
	
	The prefix may not contain any embedded numeric values. For example a 30 days arrears value in location 25 can not be referred to as
	
	`ARREARS30DAYS25`
	
	but rather as

	ARREARSTHIRTY25
location	The first data byte of a Fixed or Undefined format buffer is location 1; the first data byte of a Variable format buffer is location 5, i.e. the first byte following the Record Descriptor Word (RDW). The maximum value that can be specified is 32760 (the maximum for QSAM access), or the LRECL value of the appropriate file, whichever is smaller. In the case of Undefined format buffers, the BLKSIZE value is used, rather than the LRECL value. The advanced operand syntax allows you to specify the location of a field within a buffer dynamically via the use of SELECTIT UCTRs. Rather than explicitly defining the location in your SELECTIT command statements, you may specify these values indirectly via the UCTR variables. If UCTRx is coded for the location value, the current value in the UCTRx when the command is executed will be used as the buffer location. When location is specified dynamically via UCTRx, the prefix is required. (e.g. INPUT1:UCTRA)
/length	the length value to be used by the SELECTIT command. When a SELECTIT command has more than 1 buffer operand, only one of the operands requires a length value; it may be attached to either of the two operands. The slash (/) is required, and must be neither preceded nor followed by blanks. Note that an alphabetic prefix is not allowed for the length value. The advanced operand syntax allows you to specify the length of a field within a buffer dynamically via the use of SELECTIT UCTRs. Rather than explicitly defining the length of fields in your SELECTIT command statements, you may specify these values indirectly via the UCTR variables. If UCTRy is coded for the length value, the current value in the UCTRy when the command is executed will be used as the length operand.

Following are some examples of operand coding and how the operand is interpreted by SELECTIT.

Examples

43	A reference to location 43 in the default buffer for the operation. The 'default buffer' is determined by the context of the operand; see the description under the 'prefix' topic.
ACCTNO43	Same as above except the alpha string 'ACCTNO' precedes the value 43 to indicate the reference is to an account number field.

62/7	A reference to a 7 byte long field at location 62 in the default buffer for the operation. The 'default buffer' is determined by the context of the operand; see the description under the 'prefix' topic.
CUSTNO21/5	A reference to a 5 byte long field at location 21 in the default buffer for the operation. The alpha string CUSTNO reminds us that the field is the customer number.
INPUT2:32	A reference to a field starting at location 32 in the buffer associated with the INPUT2 file.
INPUT1:UCTRD	A reference to field within the buffer associated with the INPUT1 file. The actual location within the buffer will be determined by the contents of the UCTRD variable when the statement is actually executed. e.g. If, at execution time UCTRD=12, then the operand will reference location 12 of the INPUT1 buffer.
ADDRESS16/UCTRL	A reference to a field (an address) at location 16 within the default buffer for the operation. The length of the field will be determined at execution time by the contents of UCTRL. e.g. If UCTRL=36 then the address field will be treated as being 36 bytes long.
OBUFF:UCTRQ/ UCTRL	A reference to a dynamically located field within the common output buffer (OBUFF:). The location within the buffer will be determined by the contents of UCTRQ; the length of the field will be determined by UCTRL.

Floating Buffer Operands

The Fixed Buffer Field operands require that you know exactly where the field you want starts; this will not always be the case. For this reason, the ALTER and IF commands permit the use of 'floating' locations.

This feature allows you to scan buffers for the presence (or absence) of some data value, and optionally modify fields accordingly. The 'scan' and 'modify' features operate in a manner similar to the 'find' and 'change' commands of an editor.

This section will simply discuss the official syntax of a 'floating' operand. If you are not familiar at all with processing 'floating' fields then just skim this section for a 'feel' of the operands. An entire later section is given over to the subject of 'floating' fields and how they are handled. See "Floating Field Handling".

Syntax

```
[prefix:]0[{+adj¦-adj | +PLOC | +PLOCEND | +FLOCTERM}][/{length |
UCTRy | =}]
```

prefix:	The optional 'prefix' portion of the operand allows you to specify explicitly the buffer to which this operand refers. If not used, SELECTIT normally directs the operand to a default buffer as determined by the context of the operand as follows: **IF command operands** The operands are assumed to refer to the INPUT1 buffer. **ALTER source operand** The operand is assumed to refer to the INPUT1 buffer. **ALTER destination operand** The operand is assumed to refer to the output buffer (OBUFF). 'prefix' may be one of the Input Buffers (INPUT1: through INPUT9:), the Output Buffer (OBUFF:), or one of the User Buffers (UBUFFA: through UBUFFZ:). The colon (':') is required, and must be neither preceded nor followed by blanks.
'0' (zero)	This '0' constant acts as a trigger to SELECTIT to activate 'floating' operand processing. Note: The '0' is used as a convenient 'trigger' since '1' is the smallest possible normal value for a buffer location. It must be coded as a single '0' to be effective. Values such as `'000', '00'`, or `UCTR`x operands with a value of zero will not trigger 'floating' operand processing. **'adjusted' 0 locations** Your processing requirements may be such that you need to manipulate areas of the buffer 'near' the location which has been located by an

	'IF 0 = ...' command. The following optional values may be entered to achieve this:
+adj/-adj	This would be entered as **'0+4', '0+10', '0-3'**, etc. It indicates that the operand is to refer to the last location set by an **'IF 0 = ...'** command adjusted by the numeric value coded.
0+PLOC	The operand is to refer to the byte following last location set by a 'floating' IF. i.e. the byte following the leftmost byte of the located value.
0+PLOCEND	The operand is to refer to the byte following the last byte of located value.
0+FLOCTERM	The operand is to refer to the last location set by a 'floating' IF adjusted right by the current value of FLOCTERM. This form of operand may only be used in the form 0+FLOCTERM. as the target field of an ALTER command. It describes the location of the delimiter in the previous delimited ALTER ... COPY statement.
/length	The length value to be used by the SELECTIT command. When a SELECTIT command has more than 1 buffer operand, only one of the operands requires a length value; it may be attached to either of the two operands. The slash (/) is required, and must be neither preceded nor followed by blanks. The length value may be a: **numeric value** Note that an alphabetic prefix is not allowed for this value. **UCTRy specification** If UCTRy is coded for the length value, the current value in the UCTRy when the command is executed will be used as the length operand. **/=** The '/=' form of the length operand is used to trigger unique processing. This facility is covered fully in "Changing Strings" An example showing its use is covered by "Building commands for another program".

Keyword Variable Operands

SELECTIT has a number of predefined Keywords which may be used as command operands. Their purpose is to provide you with access to data items which are not part of the data you are processing from the input files. The Keyword variables fall into four basic types:

I/O related variables These are typically data items related to the files you are processing. They provide you with access to such items as the DSNAME of the file being processed, the current record number, the volser on which the dataset resides, etc.

Date/Time related variables These variables allow you to access the current date and time in a variety of formats. (e.g. Julian date, dd/mmm/yy, etc.)

'floating' field variables When using SELECTIT support for 'floating' fields, a number of Keywords are provided to assist you in manipulating the data. A full description of 'floating' field processing is found later under "Floating Field Handling".

Miscellaneous items Keywords are also provided to allow you to access such items as the current Jobname, Stepname, the Return code, the user portion of the SELECTIT page heading; UCTRx fields (user controlled counters); etc.

Syntax

```
[prefix:]keyword[{+adj | -adj}][/length]
```

prefix:	Certain of the keywords (the I/O related ones) are associated with a particular DDname; the 'prefix' operand allows you to specify explicitly which DDname this keyword refers to. If the 'prefix' is omitted, then INPUT1 is assumed. 'prefix' may be one of the Input DD's (INPUT1: through INPUT9:), or any user specified output DDname.
keyword	A valid SELECTIT keyword. **'adjusted' locations** Certain keywords allow you to use +/- adjustment values to allow access to areas within or near the predefined field.
/length	The length value to be used by the SELECTIT command. Most SELECTIT keywords have a predefined assumed length value and you need not specify the length. Some keywords allow the length to be overridden if required. The length value must be a valid numeric and is separated from the rest of the operand by a **'/'** with no intervening spaces.
	Full descriptions of the SELECTIT keywords and details on how and where they may be used is covered in "Predefined Keyword Variables".

Equated Operands

Some of the advanced forms of operand coding described in the previous sections can yield some long convoluted operands. e.g.

```
INPUT2:RECTYPE2/2

UBUFFA:UCTRX/UCTRL

OUTSTAT125/2
```

SELECTIT provides you with an EQUATE command to allow you to substitute your own symbolic name or abbreviation for any single string of characters. For example, the above examples might be done as:

```
EQUATE IRTYPE INPUT2:RECTYPE2/2.

EQUATE BUFFSAVE UBUFFA:UCTRX/UCTRL.

EQUATE OUTSTAT OUTSTAT125/2.
```

which would allow you to code statements such as

```
IF IRTYPE =C'A' THEN

ALTER BUFFSAVE COPY REC20.

ALTER OUTSTAT COPY IRTYPE.
```

... rather than the longer ...

```
IF INPUT2:RECTYPE2/2 = C'A' THEN

ALTER UBUFFA:UCTRX/UCTRL COPY REC20.

ALTER OUTSTAT125/2 COPY INPUT2:RECTYPE2/2.
```

Another obvious advantage to this is that is allows you to define data fields explicitly in one spot. If the definition needs to be changed, only one SELECTIT statement needs to be altered. There is no need to search a long series of statements for each reference to the altered field.

Although not recommended for this use, the EQUATE command can be used to alias any SELECTIT keywords, even the command names themselves. Some restraint should be used if you choose to do this since you may create SELECTIT programs which would be very difficult for anyone else to support. For example, the following commands

```
MARY HAD A LITTLE LAMB ITS FLEECE WAS WHITE AS SNOW.
```

would be quite valid if the following EQUATE statements had preceded it

```
EQUATE MARY IF.
EQUATE HAD RTYPE2.
EQUATE A =.
EQUATE LITTLE C'0'.
EQUATE LAMB OR.
EQUATE ITS BALANCE30.
EQUATE FLEECE >.
EQUATE WAS PL5'100'.
EQUATE WHITE THEN.
EQUATE AS SELECT.
EQUATE SNOW TODD(OUTPUT1).
```

The Advanced Commands

The following sections provide the complete descriptions of all the SELECTIT commands.

ALTER modify data in buffers or keyword variables

CLOSE close ddnames prior to normal termination

DEBUG request diagnostic output

EQUATE specify field definitions symbolically

GETNEXT specific read requests for input records

GOBACK Return from a routine

GOSUB transfer control to a routine and save return address

GOTO transfer control within the SELECTIT program

IF perform tests on data fields

OPTIONS specify processing options

SELECT request writing of output data

TITLE alter page heading data

The command descriptions themselves will not attempt to show how the various operands can be used. The other sections of this Advanced User's Guide will provide much more background on the use of the commands. As well, the Advanced User's Guide contains a very extensive selection of example runs. Some time spent examining these examples should prove very worthwhile.

ALTER

The ALTER command, whose basic structure is:

```
[label:] ALTER target-operand action source-operand.
```

is more easily be described when 'broken down' into 4 different sub-types.

Standard ALTER To alter the contents of a field in a buffer or one of the user modifiable Keyword variables. The location and length of the field may be specified directly by the operands or indirectly via the UCTRx variables.

Delimited Copy ALTER Similar to the Standard ALTER except that the length is not specified. The amount of data to be moved is determined and controlled by user-specified delimiters.

Translate ALTER Provides a character by character translation as specified by the user.

Buffer Copy ALTER To transfer the entire contents of one SELECTIT buffer to another.

Standard ALTER

Format

```
[label:] ALTER target-a action source-a.
```

Function (Standard ALTER) To alter the contents of a field in a buffer or the contents of one of the user-modifiable Keyword variables.

Operands (Standard ALTER)

target-a	This may be: ○ a field within any of the buffers (i.e. INPUT1-9, OBUFF, UBUFFA-Z, with a default of OBUFF for unprefixed fields) ○ a floating buffer field whose location has been set by the last floating IF. ○ any of the user-modifiable Keyword variables (i.e.outddn:DATALEN, HEADING, LINECTR, RCODE, UCTRA-Z, etc.). A length value may be present for the buffer fields; it would normally be omitted for the Keyword variables.
action	Specifies the type of change to make. Note that for the arithmetic operations (ADD/SUB), no notice is taken of overflow conditions. **ADD** Add source operand contents to target operand contents (binary). **ADDP** Add source operand contents to target operand contents (packed). **AND** Logical AND source operand contents with target operand contents; answer to replace the target operand. **COPY** Replace the target operand contents with the source operand contents. **CVBP** Convert source operand contents from binary to packed and replace the target operand. **CVBZ** Convert source operand contents from binary to zoned and replace the target operand. **CVPB** Convert source operand contents from packed to binary and replace the target operand. **CVPZ** Convert source operand contents from packed to zoned and replace the target operand. **CVXC** Convert source operand contents from hex to display hex and replace the target operand. **CVZB** Convert source operand contents from zoned to binary hex and

	replace the target operand. **CVZP** Convert source operand contents from zoned to packed and replace the target operand. **DIV** Divide one binary operand by another. The answer will replace the target operand. **EDITB** Convert source operand contents from binary to Edit format. Edit format is zero suppressed, comma delimited with a trailing '-' for negative values. i.e. 999,999,999-. Zero values will display as '0 '. **EDITB$** Convert source operand contents from binary to Edit$ format. Edit$ format is used for converting $ values as it assumes 2 decimal places. The result is zero suppressed, comma delimited with 2 decimal places and a trailing '-' for negative values. i.e. 999,999,999.99-. Zero values will display as '0.00 ' **EDITP** Convert source operand contents from packed decimal to Edit format. Edit format is zero suppressed, comma delimited with a trailing '-' for negative values. i.e. 999,999,999-. Zero values will display as '0 '. **EDITP$** Convert source operand contents from packed decimal to Edit$ format. Edit$ format is for converting $ values as it assumes 2 decimal places. The result is zero suppressed, comma delimited with 2 decimal places and a trailing '-' for negative values. i.e. 999,999,999.99-. Zero values will display as '0.00 ' **MULT** Multiply one binary operand by another. The answer will replace the target operand. **OR** Logical OR the source operand contents with the target operand contents and replace the target operand. **REPLACE** (synonymous with COPY) **SUB** Subtract the source operand contents from the target operand contents (binary). **SUBP** Subtract the source operand contents from the target operand contents (packed). **XOR** Exclusive OR the source operand contents with the target operand contents and replace the target operand.
source-a	This may represent: ○ a field within any of the buffers (i.e. INPUT1-9, OBUFF, UBUFFA-Z, with a default of INPUT1 for unprefixed fields) ○ a literal value ○ any of the read-only or user-modifiable Keyword variables. A '/nnn' length value may be coded for the buffer references; it must be omitted for literals, and it is optional for Keyword variables. <u>Length Values</u> Two types of length values are recognized by SELECTIT:

	○ Explicit length values are specified with '/length', '/UCTRx', or '/=', and are applicable to Buffer Fields and some Keyword variables. ○ Implicit length values are assumed for all Keyword variables and Floating ALTER target fields. Lengths for literals areeither implied from the contents or from the optional literal length subfield. The default length value for fixed buffer fields is the length value coded (or implied) for the other operand. The default length value for floating buffer fields is the length value coded (or implied) for the previous floating IF which resulted in a 'data found' condition.

Standard ALTER Notes

- When using EDITB, EDITB$, EDITP or EDITP$ make sure you allow for the trailing sign position (the ' ' or '-') and for the inserted commas and/or decimal point when specifying the length for the result field. The maximum size of the result field is 11 digits, 15 bytes. For example 99,999,999,999- in EDITB or EDITP; and 999,999,999.99- in EDITB$ or EDITP$.
- **Compression/Expansion of records** When data movement is requested with the ALTER command and different lengths are specified (or implied) for the target and source fields, SELECTIT will assume that the data in the record surrounding the target field should be shifted left or right to accomodate the lengths requested. e.g. if the target length is 4 and the source length is 2, the data to the right of the target field will be shifted 2 bytes left, 2 trailing pad characters will be added to the end of the record, and then the source field moved to the target location.

  If the reverse were true (target 2 and source 4), the data to the right of the target field would be shifted right and 2 bytes at the end of the record would be lost, then the source data would be moved to the target location. The first case above is referred to as 'compression', the second as 'expansion'.
- Length Values - Data Movement (COPY/REPLACE) Action code COPY (and its synonym REPLACE) has different interpretations of length values, depending on the function being performed:
  - If a fixed COPY is being performed, a single length value must be coded (or implied), either on the source or the target.
  - If a COPY is performed with two different length values coded (or implied), then expansion/ compression will be performed.
  - If a floating COPY (ALTER 0...) is being performed, a single length value on the second (source) operand will cause an assumed length for the first (target) operand equal to the length value on the previous true floating IF command. If the target field length value is coded as '/=', no expansion/ compression will take place. If a length value is coded explicitly on the target field, it will override the assumed value.
- Length Values - Boolean AND/OR/XOR Action codes AND, OR, and XOR must have a single (explicit or implicit) length value. An explicit length value may be attached to either or both of the operands; an implicit value will be assumed for Keyword variables, and floating buffer fields. If both operands include (or imply) a length value, the two values must be identical.
- Length Values - Binary Arithmetic ADD/SUB Both operands must have explicit (or implicit) length values for both binary arithmetic (ADD, SUB) codes. The two length values must each be in the range 1-4. Length values of 2 and 4 will cause a field to be considered signed; length values of 1 and 3 will be treated as unsigned. **Note**: Even though the two length values may be different from each other, expansion/ compression will not be performed. SELECTIT simply uses the lengths to ensure that valid signed binary arithmetic is performed.

- Length Values - Packed Arithmetic ADDP/SUBP Both operands must have explicit (or implicit) length values for both packed decimal arithmetic (ADDP, SUBP) codes. The two length values must each be in the range 1-16. If either field does not represent a valid packed decimal number, an abend S0C7 will occur.
- Length Values - Conversion CVxx and EDITxx types Both operands must have explicit (or implicit) length values for all conversion (CVxx) and (EDITxx) action codes. For CVXC, the length value for 'target-a' must be exactly double the length value for 'source-a'. For all other conversion action codes (i.e. CVxx - excluding CVXC): binary fields must have a length value in the range 1-4; packed and zoned fields must have a length value in the range 1-16. Invalid packed decimal fields specified for 'source-a' will cause abend S0C7 for action codes CVPB, EDITP and EDITP$, and unpredictable results for CVPZ. Invalid zoned decimal fields specified for 'source-a' may cause abend S0C7 for CVZB, and unpredictable results for CVZP.
- If the 'target-a' field for an EDITxx operation is too small for the value in the 'source-a' operand, the result will simply be truncated on the left.

Examples (Standard ALTER)

```
ALTER BALANCE30 COPY PL5'0'.

ALTER COL10 COPY COL20/10.

ALTER POSIT2 COPY INPUT5:POSIT3/20.

ALTER 10 AND INPUT1:10/2.

ALTER STATUSBYTE10 OR X'01'.

ALTER COL12/7 EDITP$ SALES10/3.

ALTER RCODE COPY F'4'.

ALTER RCODE COPY POS5/4.

ALTER COL10/10 COPY COL20.

ALTER OBUFF:5/8 CVXC INPUT1:CREC.

ALTER UBUFFA:10/10 CVXC INPUT2:3/5.

ALTER OBUFF:5/3 CVZP 20/5.

ALTER OBUFF:5/9 CVPZ 20/5.

ALTER UBUFFA:5/4 MULT UCTRA.

ALTER UBUFFA:6/2 DIV UBUFFA:12/1.

ALTER 0 COPY INPUT4:POSITION3/10.

ALTER BYTE5 COPY INPUT3:0+5/3.
```

Note: The last two examples use 'floating' ALTERs where the location is assumed to have been set by a prior floating IF.

```
ALTER OBUFF:5/8 COPY INPUT2:3/5.
ALTER UBUFFA:10/5 COPY INPUT2:3/10.
```

Note: The previous two examples use fixed ALTERs with expansion or compression. The first will compress the data and add three pad characters at the end of the record. The second will expand the data and truncate the last five characters of the buffer.

Delimited Copy ALTER

Format

```
[label:] ALTER target-d COPY source-d/delimiter-d.
```

Function (Delimited Copy) To alter the contents of a field in a buffer or the contents of one of the user-modifiable Keyword variables. This form of the ALTER command allows you to delimit the COPY operation, and terminate when one of the character(s) specified by 'literal-d' is found in 'source-d'.

Operands (Delimited Copy)

target-d	This represents a field within any of the buffers (i.e. INPUT1-9, OBUFF, UBUFFA-Z, with a default of OBUFF for unprefixed buffer fields). Keyword variables are not allowed; a length value must not be present.
COPY / REPLACE	Delimited copy ALTERS are only supported by the COPY / REPLACE style Alters.
source-d	This represents a field within any of the buffers. A length value must NOT be coded (rather, the length field is determined dynamically, by the value specified for 'delimiter-d').
/delimiter-d	This specifies one or more delimiters to be used in place of the normal length value to determine the number of bytes to be altered. This operand should be coded as a 'C' or 'X' format literal with one or more delimiters making up a literal string. The first occurrence of any of the delimiters in 'delimiter-d' will signal the end of the COPY operation. The source buffer will be scanned for the specified delimiters, from the specified starting position to the end of the record. If no delimiter is found, the COPY operation will not be performed. If a delimiter is found, the target field will be replaced with all bytes starting at the specified target location, up to but excluding the delimiter byte. Note that if the byte at the starting location is equal to one of the delimiter bytes, no copy will be performed.

Result Fields

If a delimiter is found, SELECTIT will update the Keyword variable FLEN with the length up to and excluding the delimiter, and set up 0+FLOCTERM to point at the location of the delimiter within the source buffer.

If no delimiter is found, FLEN and 0+FLOCTERM will remain unchanged from their prior values.

Length Values (Delimited Copy) Normal length values are not used for this format of ALTER. Instead, a dynamic length is specified by 'literal-d', which must

be attached to the second operand ('source-d').

Examples (Delimited Copy)

```
ALTER BYTE5 COPY POSITION13/C'$#@'.
ALTER 0 COPY INPUT7:5/C'$#@'.
ALTER BYTE5 COPY INPUT3:0+5/C'$#@'.
```

Note: The last two examples use 'floating' ALTERs where the floating field location was set by a previous floating IF.

Translate ALTER

Syntax

```
[label:] ALTER target-t TRANSLATE literal-t/len-t.
```

Function (ALTER Translate) To alter the contents of a field in a buffer or the contents of one of the user-modifiable Keyword variables. This form of the ALTER command allows you to replace occurrences of specific character(s) with different characters.

Operands (Translate)

target-t	This represents a field within any of the buffers (i.e. INPUT1-9, OBUFF, UBUFFA-Z, with a default of OBUFF for unprefixed fields). Keyword variables are not allowed. A length value must not be specified.
TRANSLATE	TRANSLATE requests translation of the source field, using a translate table, as described by 'literal-t'.
literal-t	This operand is coded as a literal string of byte pairs. Each pair of bytes in the literal represents an individual byte translation. The first byte of the pair represents the value to be translated, and the second byte of the pair represents the value to which it is to be translated. The literal is normally coded as either a 'C' or 'X' format literal and must specify an even number of bytes. For example, C'AaBbCc' signifies a request for translation of upper case 'A/B/C' to lower case 'a/b/c'. If more than one translation of a particular byte is present in the literal, the rightmost occurrence will be used.
/len-t	This represents the number of bytes in the target field for which translation is to be attempted. It does not have any bearing on the length of the literal being used for translation. A single length value ('/len-t') must be specified, attached to the second operand ('literal-t'). No other forms are allowed.

Examples (Translate)

```
ALTER 10 TRANSLATE C'AaBb'/5.
ALTER 10 TRANSLATE X'C08BD09B'/15.
```

Note: The last example translates braces ('squiggly brackets') from the standard EBCDIC values to their 3800 equivalents.

Buffer Copy ALTER

Syntax

```
label:] ALTER target-buffer COPYBUFF(source-buffer).
```

Function (Buffer Copy) To copy an entire buffer to another buffer.

Operands (Buffer Copy)

target-buffer	This operand defines the target buffer to which another buffer is to be copied. 'target-buffer' may be any of OBUFF, or UBUFFA-Z.
COPYBUFF(source-buffer)	This operand defines the source buffer from which the copy is to be made; COPYBUFF requests that the contents of the buffer specified by 'source-buffer' be copied to the buffer specified by 'target-buffer'. 'source-buffer' may be any of INPUT1-9, OBUFF, UBUFFA-Z; no default is supplied by SELECTIT. If the target buffer is longer than the record in the source buffer, then the target buffer will be padded on the right with the fill-character, as defined by the OPTIONS PADCHAR value, which defaults to a space (X'40'). If the target buffer is shorter than the record in the source buffer, then the record will be truncated on the right, to the length of the target buffer.
Length values (Buffer Copy)	Length values are never specified for this form of ALTER.

Record Lengths

You should be aware that the Output Buffer and all Input Buffers have implied 'record length' values which reflect the size of the current record in the buffer. A buffer copy only moves this amount of data even though the buffer itself may be much larger.

For example, an Input Buffer is large enough to hold the maximumlength record (i.e. the LRECL value), although the previous record read into that buffer may be shorter than the buffer - this situation typically occurs with RECFM=V or U. records. The actual record length is contained in the Keyword variable DATALEN.

Examples (Buffer Copy)

```
ALTER OBUFF COPYBUFF(INPUT7).
ALTER UBUFFA COPYBUFF(INPUT2).
ALTER OBUFF COPYBUFF(UBUFFB).
```

CLOSE

Syntax

```
[label:] CLOSE {INPUT(ddname) | OUTPUT(ddname)}.
```

Function The CLOSE command requests an immediate CLOSE of the specified file. You must be sure that no further references are made to this file name by your SELECTIT statements. An example using CLOSE can be found in "Controlling OPEN/CLOSE of files". If CLOSE is not used all files are closed when SELECTIT terminates.

Operands

INPUT(ddname) | OUTPUT(ddname)	This operand specifies the 'ddname' to be CLOSEd and its processing mode (input/output). **Note**: If CLOSE is used against INPUT files you must ensure that you make no further references to the data in the input buffer for that file. While data in the buffer for a closed file may seem valid, it should not be used. If you choose to CLOSE the INPUT1 file, you should specify OPTIONS OBFORCE(ON) to ensure that a separate output buffer is created.

DEBUG

Syntax

```
[label:] DEBUG [TRACE({INIT | OFF | ON})]
               [IOTRACE({INIT | OFF | ON})]
               [IODUMP({INIT | OFF | ON})] .
```

Function: The flexibility of SELECTIT, can sometimes turn into a problem. If you design and code a complex set of SELECTIT statements which produce incorrect results, how do you determine where things went wrong? The DEBUG command allows you to select a variety of additional outputs on SYSPRINT to assist you in resolving the error.

WARNING: Use of the DEBUG command may result in a very large volume of printed output. For this reason, we recommend that you use the smallest available input files, when using any of the DEBUG facilities. You should also try to be as selective as possible about where, in your logic, any of the tracing functions are turned ON or OFF.

Operands

TRACE (INIT | OFF | ON)	This operand controls the SELECTIT logic trace output. When ON, SYSPRINT will display the flow of program control through your control statements. The output will consist of multiple entries on a line, with the format: <**ssss/www**> where: **ssss** is the statement number ssss **www** is the 'word' within that statement. www
IOTRACE (INIT | OFF | ON)	This operand controls the tracing of the file Input/Output. When ON, SYSPRINT will display the file-id and the record number for all I/O activity on the input/output files as it occurs. Refer to the IODUMP operand for additional tracing capabilities.
IODUMP (INIT | OFF | ON)	This is an extension of IOTRACE which, in addition to displaying the file-id and record number, will also DUMP the records themselves.

INIT | OFF | ON Values

These values for TRACE, IOTRACE and IODUMP all work the same way. The INIT value causes tracing functions to be 'ON', before the processing of your control statements begins (i.e. it is detected at 'compile' time and is treated as a null option at execution time).

The ON and OFF values allow you to turn tracing on and off at specific points in your SELECTIT command program.

Coding Restrictions

- Although TRACE, IOTRACE, and IODUMP are all shown as optional keywords, at least one of them must be present.
- 'YES' may be used interchangeably with 'ON'; 'NO' may beused interchangeably with 'OFF'.
- When using the DEBUG facilities, a SYSPRINT DD statement must be present.
- Use of any DEBUG statements will force the use of OPTIONS TRACENUM(ON).

EQUATE

Syntax

```
[label:] EQUATE symbol string.
```

Function Specifies a 'symbol' and an associated character string. Throughout the remainder of the program, SELECTIT will substitute the character string in place of the defined 'symbol'.

Operands

'symbol'	A 1 to 8 character name to which you wish to assign a character string value.
'string'	The value which you wish to be assigned to 'symbol'. This string must follow all normal rules for a SELECTIT 'word'. Remember also that the EQUATE command follows standard SELECTIT sentence structure which states that all sentences end with a period. Therefore 'string' will always be terminated by a period. e.g. **EQUATE BUFFLOC UBUFFA:1/12.**

Notes: Symbols as 'words'
Substitution takes place only at the SELECTIT 'word' level; a 'symbol' will not be recognized unless it is a completely independent 'word'. For example:

```
EQUATE XXX YYY.
IF XXX = C'XXX' THEN ...
```

The above IF is interpreted as

```
IF YYY = C'XXX' THEN ...
```

Not as

```
IF YYY = C'YYY' THEN ...
```

Duplicate EQUATEs
If more than one EQUATE is present which defines the same 'symbol', the first one encountered is the one used. NO warning of the duplicate definition is provided.

Recursive EQUATEs
Recursive EQUATEs are not supported. A 'symbol' will be substituted with the 'string' with which it was defined, even if that string has itself been defined as some other value.

```
EQUATE AAA BBB.
EQUATE BBB CCC.
```

```
IF AAA = C'X' THEN
```

the above IF will be substituted as

```
IF BBB = C'X' THEN
```

not as

```
IF CCC = C'X' THEN
```

Program Listing
The listing, produced on SYSPRINT at runtime, will not show the actual substitutions as has been done in the examples above. Your SELECTIT statements will be listed as they are entered. To indicate that SELECTIT has recognized and acted on the symbolic substitution, the 'symbols' which have been processed will be underlined in the listing.

GETNEXT

Syntax

```
[label:] GETNEXT [FILE(ddname)] [COPYBUFF({YES | NO})] [EOF(label)] .
```

Function The GETNEXT command allows you to read the next record from an input file without having to release logical control via the implicit label RETURN. In fact, it is the only way to read records from secondary input files (INPUT2-9). The next input record is read and control is passed to the next command in sequence.

DD Statement Requirements

For each 'ddname' referenced in the SYSIN commands, there must be a corresponding DD statement in the JCL.

Operands

FILE(ddname)	The ddname of the file to be read. This must be one of INPUT1 through INPUT9. If the FILE operand is omitted, FILE(INPUT1) will be assumed.
COPYBUFF(YES | NO)	COPYBUFF(YES) indicates that the record being read should also be copied to the Output buffer (OBUFF), thus avoiding a separate command, such as **'ALTER OBUFF COPYBUFF(ddname).'** COPYBUFF(NO) indicates that the record is not to be copied to the Output buffer. If the COPYBUFF parameter is omitted, the default for INPUT1 is based on the OPTIONS AUTOCOPY setting, and the default for INPUT2-9 is COPYBUFF(NO). If COPYBUFF(YES) is in effect (explicitly or by default), and the Output buffer is longer than the Input buffer for 'ddname', then the Output buffer will be padded on the right with the character specified by the OPTIONS PADCHAR parameter, which has a default value of X'40' (a blank). For a description of AUTOCOPY see "OPTIONS".
EOF(label)	The name of a label to which control will be transferred, when the end-of-file condition is detected. If the EOF operand is omitted, the default of EOJ will be used. Note: The EOJ default can be altered through the OPTIONS EOFLABEL statement.

	Before transferring control to the EOFLABEL, SELECTIT will force the contents of the Input buffer for a file to the OPTIONS EOFFILL character, which itself defaults to high-values (i.e. all X'FF's). This 'filled' buffer is not copied to the Output buffer.

GOBACK

The GOBACK command is used to return from a routine which has been given control via a GOSUB command. Control will be given to the instruction in the program which follows the GOSUB.

Syntax

```
[label:] GOBACK.
```

Function Returns to the SELECTIT statement which follows the GOSUB which passed control to this routine. If GOBACK is issued, and no prior GOSUB has been issued, the step will terminate with appropriate diagnostics.

Operands The command has no operands.

Note: Where nested GOSUBs have been used, GOBACK returns to the return point associated with the most recently executed GOSUB.

GOSUB

The GOSUB command is used to transfer control to another command in the SELECTIT command stream and save a return address so that the routine which is given control can return to the statement following the GOSUB.

Syntax

```
[label:] GOSUB label.
```

Function Transfers control to the specified label, which must be a user defined label, and sets up a return address which may be used by an associated GOBACK command to return processing to the statement following the GOSUB command.

Operands

label	This is the label identifying the routine which is to be given control. Refer to "Basic Syntax Rules" for valid label syntax.

Usage Notes.

The routine given control via a GOSUB MUST return via a GOBACK. If the routine exits in some manner (e.g. GOTO RETURN) without issuing the GOBACK your program will eventually exceed the maximum nesting level (see below) and terminate abnormally.

The special labels RETURN and EOJ. can not be used.

GOSUBs may be nested up to a maximum nesting level (50). i.e. a routine which has been given control via a GOSUB, may itself pass on control to other routine(s) via GOSUBs.

Example

```
IF STATFLAG3 = C'G' THEN GOSUB GROUP.
IF STATFLAG3 = C'C' THEN GOSUB CORPORATE.
GOTO RETURN.
GROUP: *: PROCESS GROUP RECORDS.
.
.
GOSUB COMMON.
GOBACK.
CORPORATE: *: PROCESS CORP RECORDS.
.
.
GOSUB COMMON.
GOBACK.
COMMON: *: DO COMMON PROCESSING.
```

```
.
.
GOBACK.
```

GOTO

Syntax

```
[label:]  GOTO  label.
```

Function Transfers control to the specified label, which may be a user-defined label, or one of the special labels RETURN and EOJ. Refer to "Basic Syntax Rules" for valid label syntax.

Special Labels

Two special reserved labels are automatically created by SELECTIT.

RETURN	A transfer to this label will terminate processing of the current input record. If the record has not already been SELECTed by some other control statement, it will not be processed any further. Processing will continue at the start of your SELECTIT commands with the next input record from INPUT1.
EOJ	A transfer to this label will immediately terminate the program even if all requested functions are not complete. Refer to the topic 'Normal Termination Rules' for more information as to how SELECTIT determines when to terminate.

Note that an error condition will be raised if you explicitly code labels with the name RETURN or EOJ in your SYSIN stream.

In order to avoid possible confusion, we recommend that you do not use any of SELECTIT's reserved words as labels.

IF

Syntax

```
[label:] IF operand-a test operand-b

   [{AND | OR} [operand-aa] [test] operand-bb] ...

   [THEN] {ALTER statement.    |
           DEBUG statement.    |
           GETNEXT statement.  |
           GOTO label.         |
           SELECT statement.   |
           DO. statement(s). END.}

   [ELSE  {ALTER statement.    |
           DEBUG statement.    |
           GETNEXT statement.  |
           GOTO label.         |
           SELECT statement.   |
           DO. statement(s). END.}
```

Function To test the contents of field(s) within buffers (i.e. INPUT1-9, OBUFF, UBUFFA-Z), or the contents of Keyword variables, for specified values, and conditionally perform other commands. An optional ELSE may be provided, to perform an alternative set of commands when the condition(s) being tested evaluate to 'false'.

Operands

operand-a/-aa operand-b/-bb	Each of these may represent: ○ a field within any of the buffers (i.e. INPUT1-9, OBUFF, UBUFFA-Z, with a default of INPUT1 for unprefixed fields) ○ a literal value ○ a Keyword variable Refer to "Advanced Syntax Rules" for coding format. A length value may be present on either operand (buffer fields). The length value must be omitted for literals and is optional for Keyword variables. Note: If /UCTRx is used as a length field, it should only be used on operand-b/-bb.
test	Specifies the test to be made. It may be either an abbreviation or a symbol as follows: **EQ = Equal to**

	GE >= Greater than or equal to GT > Greater than LE <= Less than or equal to LT < Less than MIX Logical test that all selected bits are not 'on' nor are they all 'off'. See below. NE ¬= <> Not equal to NGE ¬>= Not greater than or equal to (same as LT) NGT ¬> Not greater than (same as LE) NLE ¬<= Not less than or equal to (same as GT) NLT ¬< Not less than (same as GE) OFF Logical test that all selected bits are 'off'. See below. ON Logical test that all selected bits are 'on'. See below. MASK Character mask (partial equality - see below)

IF Notes

Pattern Matching (Partial Equality) MASK

You can use a pattern for matching, rather than an exact literal, by specifying MASK rather than EQ (or '='). The 'oper' value MASK may only be used with a 'C' or 'X' format literal as the second operand, and functions in a similar manner to EQ (=), except that the literal may contain one or more question marks (C'?' or X'6F'), indicating a 'wild-card' byte.

For example, C'??ABC' indicates a 5-character string, where the first two bytes may be any characters, but bytes 3-5 must be 'ABC'. Leading '?' values may not be used if a 'floating' comparison ('IF 0 MASK ...') is being used.

C'A??BC' would be a valid literal for both regular and 'floating' comparisons.

Bit Testing MIX/OFF/ON

The three 'oper' values MIX, OFF, and ON, each test specific bits within a single byte. Each operates like a TM (Test under Mask) instruction, and may only be used with a literal as the second operand. If more than one byte is specified in the coded literal, only the first byte will be used; in addition, only the first byte of the first operand will be tested.

Within the single byte of the second operand, a '1' bit represents a bit position that is to be tested, and a '0' bit represents a bit position which is not significant to the test. A test is considered 'true' as follows:

ON For each '1' bit in the literal (operand-b), there must be a '1' (ON) bit in the corresponding position of operand-a.

OFF For each '1' bit in the literal (operand-b), there must be a '0' (OFF) bit in the corresponding position of operand-a.

MIX The literal (operand-b) must contain at least 2 '1' bits. Each corresponding bit position in operand-a will be tested; there must be at least one '1' (ON') and at least one '0' (OFF) in the tested bit positions.

Examples MIX/ON/OFF

For example, a single byte containing the value X'A3' (binary 10100011) is tested as follows:

ON X'80' True (binary 10000000)
ON X'40' False (binary 01000000)
ON X'23' True (binary 00100011)
ON X'14' False (binary 00010100)

OFF X'80' False (binary 10000000)
OFF X'40' True (binary 01000000)
OFF X'23' False (binary 00100011)
OFF X'14' True (binary 00010100)

MIX X'80' False (only one bit tested)
MIX X'40' False (only one bit tested)
MIX X'A0' False (both bits tested are on)
MIX X'50' False (both bits tested are off)
MIX X'90' True (binary 10010000)
MIX X'FF' True (binary 11111111)

Floating Fields
Floating fields are specified by using a location value of 0 (possibly adjusted by a numeric value, PLOC, PLOCEND, or FLOCTERM), and are used to scan across a record, rather than testing a fixed location.

The only valid 'oper' values for floating comparisons are 'EQ', 'MASK', and 'NE'. Full explanations of processing with Floating fields are provided in "Floating Field Handling".

AND/OR
Multiple tests may be made in any AND/OR relationship. The AND/OR relationships are treated in the same manner as COBOL IF statements (i.e. AND takes precedence over OR, otherwise left to right). Parentheses may not be coded to override this hierarchy. A maximum of 1023 conditions may be specified; if you code in excess of this number, a compile-time error will occur.

Implied location/operator
If multiple tests are being made against a field, the first operand may be omitted if it is the same as the one in the preceding test; both the first operand and the operation may be omitted, if both are identical to those in the previous test. The following are all valid ways of specifying the same test:

```
IF P10 > C'A' AND P10 < C'Z' OR P10 = C'.'
IF P10 > C'A' AND < C'Z' OR P10 = C'.'
IF P10 > C'A' AND < C'Z' OR = C'.'
```

Similarly:

```
IF P10 = C'A' OR P10 = C'B' OR P10 = C'C'
IF P10 = C'A' OR C'B' OR C'C'
```

And:

```
IF P10 ¬= C'A' AND P10 ¬= C'B' AND P10 ¬= C'C'
IF P10 ¬= C'A' AND C'B' AND C'C'
```

Note the use of AND in conjunction with '¬='; this is the inverse of the previous example, which used OR and '='.

If 0+PLOC or 0+PLOCEND is used as an implied subject, then each 'true' comparison will cause the starting point for subsequent comparisons to be updated.

If '0', '0+nnn' or 0+FLOCTERM is used as an implied subject, the scan will always start at the same fixed location.

If you wish to use /UCTRx as a length value in conjunction with implied subjects, it must be attached to operand-b/-bb; it will not be remembered for the second and subsequent comparisons.

THEN command ...
The command to be executed (i.e. ALTER, DEBUG, DO, GETNEXT, GOTO, or SELECT) if a true result occurs, is coded exactly the same as the unconditional form of the command. Note that IF, OPTIONS, RUN, and TITLE are not permitted. To improve readability of the IF statement, the word THEN may be placed before the action to be taken, but is not required; for example, the two statements below are identical in function:

```
IF 20 = C'X' THEN GOTO RETURN.
IF POS20 = C'X' GOTO RETURN.
```

ELSE command ...
You may optionally specify an ELSE command, which will be executed when the condition being tested is false. The available commands, and the syntax, are identical to the 'true' sequence shown above. If ELSE is coded, the period following the THEN command is still required, and ELSE must be the next command following either the THEN command, or the END statement in a THEN DO/END group.

THEN and ELSE DO/END Groups
It is possible to specify a series of commands to be executed after either or both of THEN and ELSE. This is accomplished through the use of a DO/END group. If DO is coded as the resulting action for THEN or ELSE, all commands following DO, up to and excluding the next END command (exception: see below - 'Nested IF'), will be executed.

A DO/END group may contain labelled statements and/or other IF statements (which themselves may contain DO/END groups). A GOTO may also be included within the group; if it causes a branch outside the group, control may not be returned properly.

Note that a GOTO from outside the range of the DO/END to a labelled statement within the range will cause the labelled statement and all others prior to the END to be executed, followed by the statements immediately following END. **Note**: A null DO/END group is permitted under either THEN or ELSE.

Nested IF
A DO/END group may include nested IF commands. When a nested IF itself contains a DO/END group, each END statement will be paired with the closest preceding unmatched physical DO in the SYSIN stream.

Examples

```
IF ACCTBAL33 > PL4'100' THEN GOTO RETURN.
IF COLUMN20 = C'XX' ALTER COL20 COPY C'44'.
IF STATUS10 = C'A' OR C'B' OR C'C' GOTO RETURN.
IF 20 GT P'50' AND 30 NE C'X'
```

```
OR 50 EQ X'F3CE' THEN SELECT TODD(OUTPUT2).
IF BYTE30 ON X'80' GOTO FLAGGED.
IF CREC > F'1000' AND STATUS10 = C'A' GOTO COPYIT.
IF RCODE > F'4' THEN GETNEXT.
IF COL20 = COL40/5 GOTO EQUAL.
IF INPUT2:LOC20 = INPUT5:LOC20/5 GOTO MATCH.
IF INPUT3:30 < INPUT4:20/6
THEN ALTER 20 COPY INPUT4:20/6.
ELSE ALTER 20 COPY INPUT3:30/6.
IF INPUT2:1 MASK C'??AB?C' GOTO RETURN.
IF 0 = C'DSN=' THEN GOTO DSNFOUND.
IF 0+4 = C'SYS1' THEN GOTO SYS1X.
IF 0+PLOC = C'SYS1' THEN GOTO SYS1X.
IF 0 NE C'ABC' THEN GOTO NOABC.
IF INPUT3:0 MASK C'AB???C' GOTO RETURN.

IF COL20 EQ C'BAD' THEN DO.
   ALTER COL20 COPY C'GOOD'.
   ALTER COL30 COPY C'00000'.
   ALTER COL40 COPY CL20' '.
END.
```

The next example shows the use of Nested IFs and labelled statements within DO/END groups:

```
IF COL20 EQ C'BAD ' THEN DO.
   ALTER COL20 COPY C'GOOD'.
   ALTER COL30 COPY C'00000'.
   IF COL40 EQ C'BAD ' THEN DO.
      ALTER COL40 COPY C'GOOD'.
      ALTER COL50 COPY C'00000'.
      GOTO ALTERED.
   END.
   ALTER COL60 COPY 20C' '.
   ALTER COL80 COPY 10C'-'.
   ALTERED:
   ALTER COL90 COPY C'ALTERED'.
END.
```

The next example shows an alternative method of coding, using ELSE:

```
IF COL20 EQ C'BAD ' THEN DO.
   ALTER COL20 COPY C'GOOD'.
   ALTER COL30 COPY C'00000'.
   IF COL40 EQ C'BAD ' THEN DO.
      ALTER COL40 COPY C'GOOD'.
      ALTER COL50 COPY C'00000'.
   END.
   ELSE DO.
      ALTER COL60 COPY 20C' '.
      ALTER COL80 COPY 10C'-'.
   END.
   ALTER COL90 COPY C'ALTERED'.
END.
```

OPTIONS

The OPTIONS command is used to specify alternative values for some of the SELECTIT defaults. A full description of the usage and need for the OPTIONS command is in "Customizing the Run Time Environment", this section covers only the syntax of the command.

Syntax

```
[label:] OPTIONS option-keyword(value) ... .
```

Placement The OPTIONS statement(s) must be the first statement(s) in the SYSIN file. i.e. they must precede any executable statements. Comments and TITLE statements may precede OPTIONS.

Operands The OPTIONS statement may specify any or all of its optional keywords on one statement. Options which are not specified are not altered. If you desire, multiple OPTIONS statements may be used but they must all be placed together at the beginning of your SELECTIT control statements. In all cases, the default setting is shown first. For all parameters which have a choice of 'ON' or 'OFF', 'YES' may be used interchangeably with 'ON', and 'NO' may be used interchangeably with 'OFF'.

The valid option keywords and their defaults are:

Keyword	Default
AUTOCOPY(ON | OFF)	ON
CAPS(ON | OFF)	ON
CBLKSIZE(ON | OFF)	ON
DEFER(ddname)	none
EOFFILL(X'FF' | 1-byte-literal)	X'FF'
EOFLABEL(EOJ | label)	EOJ
LIST(ON | OFF)	ON
MEMBSTAT(OFF | ON)	OFF
OBFORCE(OFF | ON)	OFF
OPCHECK(ON | OFF)	ON
PADCHAR(X'40' | 1-byte-literal)	C' ' (blank)
PDSMODE(OFF | ON)	OFF
RETLABEL(RETURN | label)	RETURN
STATS(ON | OFF)	ON
TRACENUM(ON | OFF)	ON
UBSIZE(1024 | numeric)	1024

User PARM field data

It is possible to pass data to the SELECTIT program from the PARM field of the EXEC statement which invokes

SELECTIT. This is similar to the way OPTIONS operands may be passed via the PARM field.

Because of the internal methods used, the PARM keyword could also be legally coded on an actual OPTIONS statement although it would be illogical to do so. For details on the User PARM support, see "User PARM Data".

SELECT

Syntax

```
[label:] SELECT [RECS({ALL | nnn})]
                [LENGTH(nnn)]
                [SREC(sss)]
                [BY(bbb)]
                [SKEY(literal)]
                [EKEY(literal)]
         [THEN] [TODD(ddname)]
         [THEN] [PRINT({CHAR | DUMP | HEX})]
         [THEN] [GOTO(label)] .
```

Function SELECT controls the writing of records from the Output buffer, specifies selection criteria (other than those provided by IF), and describes the actions to be performed if the criteria are met.

SELECT operates at the record level, as opposed to IF, which operates at the field level.

Coding Restrictions
Although TODD, PRINT, and GOTO are all shown as optional keywords, at least one of them must be present. Refer to 'Action Commands' below for details.

If SYSIN is present, at least one SELECT statement with either the TODD or PRINT operand (or both) is required, unless OPTIONS OPCHECK(OFF) is in effect.

DD Statement Requirements
If the TODD operand is coded, there must be a DD statement present for 'ddname', even if the SELECT statement is part of a conditional IF statement. i.e. the fact that the SELECT statement may never get executed is immaterial.

If the PRINT operand is coded, SYSPRINT (which is normally optional) becomes mandatory.

Operands

RECS(ALL) RECS(nnn)	Any numeric value or ALL. Specifies the number of records (nnn) this SELECT is to process. If the RECS operand is omitted, the default is RECS(ALL). If 'nnn' is greater than the number of records in the input file, it will be treated as though ALL had been specified. **Note**: The RECS(nnn) count is matched against the number of times that this SELECT command has been executed, not the number of records read from Input files. See also the discussion below on the effects of SKEY/EKEY.
LENGTH(nnn)	Any numeric value. Specifies the length of the record to be written or printed. If the LENGTH operand is omitted, the default length is based on the last record copied to the output buffer from an input buffer (the length is not affected by

	copying a record from a User Buffer). You may also control the length of a record by ALTERing the Keyword variable DATALEN, or (in the case of Variable-length INPUT1 records) by altering the RDW in bytes 1 and 2 of the Output buffer. The LENGTH value takes precedence over DATALEN and the RDW field. LENGTH is only valid with TODD and PRINT; it is ignored if the only 'action command' is GOTO. For TODD, LENGTH is ignored if the RECFM of 'ddname' is F. The LENGTH operand may be used on a SELECT PRINT(nnn) to cause only the leading 'nnn' bytes of the record to be printed. e.g. if you are processing 23000 byte records and the information you wish to print is all within the 1st 100 bytes of the record, a command such as **`SELECT PRINT(DUMP) LENGTH(100).`** will save you a considerable volume of printout.
SREC(sss)	Specifies the starting record number (sss) of the primary input file (INPUT1), at which this SELECT will start processing records. If the SREC operand is omitted, SREC(1) will be assumed.
BY(bbb)	Specifies that only every bbb'th record is to be processed by this SELECT. If the BY operand is omitted, BY(1) will be assumed (i.e. bbb=1). SREC/BY Usage The record numbers selected will be sss, sss+bbb, sss+2*bbb, ... The SREC and BY operands will normally be used for non-keyed files. If either operand is specified when INPUT1 is a keyed file, and the SKEY and/or EKEY parameter is also specified, the function will be unpredictable, since the match will be against the number of records actually read from INPUT1, not the record number of the record within the file. Note: The SREC and BY counts are matched against the number of times that this SELECT command has been executed, not the number of records read from Input files. See also the discussion which follows on the effects of these operands with SKEY/EKEY.
SKEY(literal)	The start key at which this SELECT is to start processing. If omitted, the default is SKEY(low-values) (i.e. all X'00').
EKEY(literal)	The end key at which this SELECT is to terminate processing. If omitted, the default is EKEY(high-values) (i.e. all X'FF').

	SKEY/EKEY Usage The SKEY and EKEY operands are only valid when INPUT1 is a keyed file. If the SKEY and/or EKEY parameter is specified for a non-keyed file, an error message will be produced to that effect. See the discussion below for additional details on the effect of SKEY and EKEY on the GETNEXT command.
TODD(ddname)	Requests that the record currently in the Output buffer be written to the named output file 'ddname'.
PRINT(....)	Requests that the record currently in the Output buffer be printed on SYSPRINT. **(CHAR)** character only is printed. **(DUMP)** both character and hexadecimal are printed. **(HEX)** hex only is printed.
GOTO(label) | GOTO label	Following any other requested actions (i.e. TODD and/or PRINT) for the SELECTed records, control is passed to the named label. Note that the parentheses around the label are optional.

Action Commands

A SELECT statement may contain any combination of TODD, PRINT, and GOTO action commands, but not more than one of each. It must contain at least one of them. It is perfectly valid to have a SELECT with just a GOTO. e.g. if you wished to process every other record

```
SELECT BY(2) GOTO RETURN.
ALTER ....
SELECT TODD(...).
```

The PRINT, TODD, and GOTO commands may be coded in any order, but PRINT and TODD will be processed prior to the GOTO (if one is present).

THEN To improve readability of the SELECT statement the word THEN may be inserted before any action; for example:

```
SELECT RECS(20) TODD(OUTFILE)
THEN PRINT(CHAR)
THEN GOTO REFORMAT.
```

Keyed Files - Effect of SKEY/EKEY on GETNEXT

SKEY and EKEY are used by SELECTIT to determine the lowest and highest keys to be used on GETNEXT processing (this includes the automatic GETNEXT done at RETURN). The lowest SKEY value encountered on all SELECT statements in SYSIN will be the first keyed record read from the INPUT1 file; similarly, the highest EKEY value will determine the last record read.

If multiple ranges of keys are present on SELECT statements, then some records with keys falling between the lowest SKEY and the highest EKEY may not be read.

Example (using 1-byte keys):
There are 2 SELECT statements in SYSIN, where one has **`'SKEY(C'B') EKEY(C'D')'`** and the other has **`'SKEY(C'M') EKEY(C'Q')'`**.

INPUT1 consists of 26 records with key values C'A', C'B', C'C', ..., C'Z'. The only records read from INPUT1 will be the 3 records with keys 'B' to 'D' and the 5 records with keys 'M' to 'Q'; the remaining 18 records will not be read, and will therefore never be processed by the SELECT statements.

For this reason, RECS, SREC and BY should be used with caution when processing keyed files. In the example above, the record with key 'C' will be the second record processed by SELECT, even though it is the third record in the input file.

Active/Complete SELECTs

IMPORTANT: When there are no 'ACTIVE' SELECT statements left, SELECTIT will terminate, even though all the records from the INPUTn files may not have been read. i.e. SELECTIT will terminate before end of file when all RECS(nnn) are satisfied.

If OPTIONS OPCHECK(OFF) is in effect, then the absence of 'ACTIVE' SELECT statements will not cause termination.

TITLE

Syntax

```
[label:] TITLE 'desired title characters'.
```

Function Places your data in the SELECTIT page headings, in order to aid in identifying file dumps, etc. If no TITLE statement is present, then the data set name of the INPUT1 file will be used. If INPUT1 references a concatenation, the first data set name in the concatenation will be used.

The Keyword variable HEADING may also be used to supply a TITLE value.

DD Statement Requirements
TITLE is only meaningful if a SYSPRINT DD statement is present. However, no error is reported if TITLE is used and no SYSPRINT is present.

Limitation A maximum of 54 characters may be specified. If more than 54 characters are coded, it will be truncated (no warning message will be produced).

Placement The TITLE command should be placed first in the SYSIN stream. If other commands precede the TITLE command, TITLE will become effective on the next output page only. If more than one TITLE is present, the most recently encountered TITLE text will be printed at each page break.

Note: The current value of the title is available in the Keyword variable HEADING.

Advanced Topics

Logic Flow

Although the SELECTIT command statements contain all the necessary components to completely control its functions, i.e.

- Reading record(s)
- Testing the contents of the record(s)
- Unconditional or conditional modification of record(s)
- Writing record(s)
- Termination

your command statements do not execute independently of SELECTIT's own main processing loop.

An understanding of where your SELECTIT command statements 'fit in' is particularly needed once you begin to exploit SELECTIT's advanced features.

Where Your Commands 'fit in'

When processing with only the primary input file (INPUT1), you do not normally read records. Think of your control statements as executing within a read loop, which reads the primary input (INPUT1) file.

```
RETURN:  read next record into input buffer
         at end of file, go to EOJ.
         copy record to the output buffer.

      >> your control statements here <<

         go to RETURN.
EOJ:     termination processing.
```

Note the following:

- If you allow your logic to 'fall out the bottom' it is equivalent to a 'GOTO RETURN.' statement.
- Your statements are given control at the 1st statement for every record read by SELECTIT at the RETURN: label. If portions of your SELECTIT statements are considered 'initialization' and are only to be executed once, then place them within a conditional IF which tests for the first record. e.g.

```
IF CREC = F'1' THEN DO.
  .
  initialization statements
  .
END.
```

Detailed Logic Flow

The following section describes the logic flow of a complete SELECTIT step execution. The flow can be broken down into five major portions:

1. Initialization and Compile

2. Start of Loop
3. User Control Statement Execution
4. End of Loop
5. Program Termination

SELECTIT processes as follows (RETURN and EOJ are implicit labels, and you must not code them explicitly as labels in the SYSIN stream):

```
(Initialization)
  Open SYSPRINT if present
  Open INPUT1 if present (Note 1)
    - Initialize INPUT1:CREC=0
    - Initialize RCODE=0
  'Compile' PARM/SYSIN statements (or SELECTIT default)
  Each statement will be processed (Note 2) as follows:
    - List statement on SYSPRINT if available
    - Open any DDname referenced by the statement, and
      initialize ddname:CREC=0
    - Detect any conditions requiring separate Output Buffer (Note 3)
  If any compile errors, go to EOJ, with abend U0012
  Create separate Output Buffer if required
  If INPUT1 absent, go to 'User control'

(Start of loop)
  RETURN:
  Read primary input record (INPUT1)
  If end-of-file:
    Move eof-fill characters to Input buffer
    see EOF buffer fill (Note 5)
    Go to EOJ (or OPTIONS EOFLABEL value) (Note 4)
  Copy primary input record to Output buffer
  (unless OPTIONS AUTOCOPY(NO) in effect)
  INPUT1:CREC = INPUT1:CREC + 1

(User control)
  Process user's statements or SELECTIT default

(End of loop)
  Go to RETURN (or OPTIONS RETLABEL value)

(Termination)
EOJ:
  Print I/O statistics and non-zero UCTRx values
  Close files and exit with RCODE as the return-code.
```

Note 1 (INPUT1 initialization)
When INPUT1 is omitted, the implicit GETNEXT at label RETURN will be bypassed the first time through ONLY, and your SYSIN statements must provide all required actions. In particular, you must never allow control to 'fall through' the bottom of the SYSIN stream, unless you have provided an OPTIONS RETLABEL value. If you do not provide a RETLABEL value, and control does fall through, then an error will occur trying to read past end-of-file from INPUT1 at RETURN.

When INPUT1 is empty, and you want the SYSIN statements to be given control regardless, you must provide an OPTIONS EOFLABEL(xxx) statement, where 'xxx' is a label within your SYSIN statements. If you do not provide such a label, then an immediate end-of-job exit will be taken at label RETURN, and your SYSIN statements will NOT be given control at all (they will still be 'compiled' though).

In addition, you must provide a RETLABEL value, in order to prevent a 'read past end-of-file' error if control is allowed to pass to RETURN.

Note 2 ('Compile' Phase)
SELECTIT makes one pass of the SYSIN stream, in order to create 'pseudo-code' for subsequent execution. The 'compile' phase will flag any syntax errors, and determine if any required DD statements are missing. If any errors (other than warnings) are detected, SELECTIT will exit immediately, with abend U0012.

Note 3 (Output Buffer/DEBUG (INIT))
If all SYSIN statements are syntactically correct, SELECTIT willthen determine if any conditions require a separate Output buffer. Further details on buffer sharing are in "Shared Input/Output Buffer".

Note 4 (Input file processing)
This section of logic is equivalent to the SELECTIT command:

```
GETNEXT FILE(INPUT1) COPYBUFF(xxx) EOF(yyy).
```

where 'xxx' is YES or NO depending on the OPTIONS AUTOCOPY setting (default ON), and 'yyy' is the label defined by the OPTIONS EOFLABEL value (default EOJ).

Note 5 (EOF buffer fill)
The end-of-file record fill character defaults to high-values (X'FF'), but may be overridden by the OPTIONS EOFFILL parameter.

How end-of-job is determined

SELECTIT uses the 'ACTIVE' status of all SELECT statements to determine when to terminate processing. Basically, SELECTIT treats a SELECT statement as either 'ACTIVE' or 'COMPLETE'. A SELECT which is coded with RECS(ALL) will always be 'ACTIVE', whereas a SELECT which is coded with RECS(10) will only be 'ACTIVE' until it has processed 10 records. It will then become 'COMPLETE'.

A SELECT which has become 'COMPLETE', logically 'disappears' and is treated as a NULL statement for the rest of the processing. When there are no 'ACTIVE' SELECT statements left (indicating no more output is to be created), SELECTIT will terminate, even though all the records from the INPUT files may not have been read.

This detection of 'ACTIVE'/'COMPLETE' SELECT statements normally serves very well as a method of controlling termination since it avoids the reading and processing of input records once all possibility of producing output records is past.

The ability to suppress this logic is provided via the OPTIONS OPCHECK(NO) statement. An example that might require this is one in which a file is read and a variety of statistics are accumulated regarding the file in the UCTRx counters. In a case such as this, no records are output at all; there is probably not even a single SELECT statement in the program, yet the entire input file must still be processed. See "Using UCTRS to accumulate statistics" for an example run.

Loop protection SELECTIT also provides you some protection from accidental coding errors in your control statements which cause a never ending loop. SELECTIT will detect the loop and force an abnormal termination of the step. Diagnostic data will be provided which describes the location of the loop in your control statements and a dump of the data record being processed at the time.

Buffers

Since SELECTIT is capable of processing multiple input and output files, a description of SELECTIT's Input, Output, and User Buffer handling is presented here.

The following toics are covered:

What do I need to know?

In most cases you need to know very little of the details of how SELECTIT controls the various buffers for you. Certain types of activity you may request do require you to have some understanding of the way data in the buffers is manipulated. You should review the material following if:

- you are examining or modifying the data in variable length records.
- you are examining or modifying the data while converting records from one record format to another. (e.g. fixed to variable, undefined to fixed, variable to undefined, etc.)
- you are altering the lengths of records.
- you are processing multiple input files which have differing record formats.

You should find that the default actions and processes used by SELECTIT simplify any actions required of you; but you must be aware of what these default processes are.

Throughout the rest of this chapter, all 'buffer' references will use the internal name of the buffer (i.e. INPUT1-9, OBUFF, and UBUFFA-Z, as explained below).

Input Buffers (INPUT1-9)

Input Buffer Description

Each input file has its own Input Buffer. Data in the Input Buffer is normally not modified in the input buffer, however it is possible to do so.

The names of the Input Buffers are identical to their DDnames, i.e. INPUT1 through INPUT9. Fields within Input Buffers are referenced by 'INPUTn:location'. Keyword variables relating to INPUT1-9 are also prefixed by 'INPUTn:'. For example, INPUT2:CREC refers to the current record number of the INPUT2 file.

Input Buffer Length and Data Length
The length of an Input Buffer is determined by the DCB LRECL value (RECFM=F or V) or the DCB BLKSIZE value (RECFM=U) of the file being read. If INPUTn references a concatenation of data sets, then the length of the Input Buffer will be recalculated for each data set in the concatenation.

In addition, there is a 'data length' associated with the record currently in the Input Buffer. This is the actual length of the last record read into the Input Buffer; the value is stored in the Keyword variable 'INPUTn:DATALEN'. For RECFM=F, the 'data length' is the same as the buffer length; for RECFM=U or V, the 'data length' may be less than the buffer length.

Since length values are not specified in buffer copies, the input 'data length' will be used to determine whether truncation or padding is to be performed.

If an ALTER to a field within an Input Buffer results in expansion, the resulting record will be truncated to the 'data length' (any data shifted out of the record will be discarded); no change in the 'data length' itself will occur, and no modifications will be made to the RDW field in a V-format Input Buffer.

Output Buffer (OBUFF)

Output Buffer Description
All output files share a single common Output Buffer. The name of the Output Buffer is OBUFF. Fields within the Output Buffer are referenced by 'OBUFF:location'. Note: Prefixed Keyword variables which apply to output DDnames are referenced by 'outddname:keyword'.

Normally, all buffer modifications made by ALTER commands are made to this buffer. All output files (via SELECT TODD...) or dumps (via SELECT PRINT...) are written from this buffer. The only time that SELECTIT copies any data automatically to the Output Buffer is when a read is performed against the primary input file (INPUT1), and OPTIONS AUTOCOPY(ON) (the default) is in effect.

If data from secondary input files (INPUT2-9), or from INPUT1 with AUTOCOPY(OFF) in effect, has to be copied to the Output Buffer, you must explicitly request such a move with one of:

```
'ALTER OBUFF COPYBUFF(INPUTn).', or
'GETNEXT FILE(INPUTn) COPYBUFF(YES) ...'
```

If data from any of the User Buffers (UBUFFA-Z), has to be copied to the Output Buffer, you must explicitly request such a move with:

```
'ALTER OBUFF COPYBUFF(UBUFFx).'
```

Output Buffer Format
The format of the Output Buffer (i.e. RECFM) is the same as the RECFM of INPUT1, regardless of the format of any Output files. In other words, a Fixed or Undefined record contains only data, whereas a Variable record contains a 4-byte RDW at the beginning of the Output Buffer. The record format (RECFM) of the record to be written is only determined when a SELECT TODD(...) statement is encountered.

If INPUT1 references a concatenation, then only the first data set in the concatenation will be used to determine the format of the Output Buffer.

If INPUT1 is omitted, the format of the Output Buffer is considered to be Fixed.

Output Buffer Length
All references to LRECL in the following discussion apply to RECFM=F or V files; when dealing with RECFM=U files, LRECL should be read as BLKSIZE.

The length of the Output Buffer will be the greatest LRECL found for any input or output file. Note that the input and output files must be referenced by a SYSIN command (INPUT1 is always referenced implicitly); it is not sufficient to just include a data set with a large LRECL in the JCL. SYSPRINT is not considered to be an output file for the purposes of this calculation.

Note: When any of the INPUTn files reference a concatenation of data sets, then the first data set in each concatenation will be used in the initial calculation. If the second (or subsequent) data set in a concatenation is found to have an LRECL value larger than the size of the Output Buffer, SELECTIT will dynamically increase the size of the Output Buffer on the first GETNEXT performed against the data set. If such a dynamic increase in size occurs, any data in the previous Output Buffer will be preserved in the newly created Output Buffer, with PADCHAR padding on the right. However, the format of the Output Buffer will remain unchanged.

Output Buffer - Data Length
The 'data length' of the record currently in the Output Buffer is initially set to the 'data length' associated with the Source buffer of the most recent buffer copy performed, either by 'ALTER OBUFF COPYBUFF(buffer)' or by GETNEXT with COPYBUFF(YES) in effect. For copies from Input Buffers, the initial value will be equal to INPUTn:DATALEN; for copies from User Buffers, the initial value will be the smaller of the UBSIZE value and the buffer length of the Output Buffer.

When an ALTER statement results in compression or expansion of the record in the Output Buffer, the 'data length' is adjusted accordingly. However, the RDW field in a V-format Output Buffer is not affected by compression or expansion.

User Buffers (UBUFFA-Z)

User Buffer Description
There are up to 26 User Buffers available, named UBUFFA through UBUFFZ. Fields within User Buffers are referenced by 'UBUFFx:location'. Only those user buffers explicitly referenced by SYSIN commands will be actually allocated.

WARNING: The contents of the User Buffers when first referenced are unpredictable. It is your responsibility to move the data you require (including any initial value) into them.

User Buffer Length and Data Length
The length of each User Buffer defaults to 1024, but may be made longer or shorter with the OPTIONS UBSIZE parameter. The 'data length' of a record in a User Buffer is equal to the buffer length.

When a buffer copy is performed from either an Input Buffer or the Output Buffer to a User Buffer, the 'data length' associated with the source will be used to determine whether truncation or padding is required.

If an ALTER to a field within a User Buffer results in expansion, the resulting record will be truncated to the length of the buffer; no modifications will be made to a possible RDW field within the first four locations of the buffer.

Moving data between buffers

Buffer Copies - INPUTn to OBUFF
When copying records from an Input Buffer to the Output Buffer, location 1 of the Input Buffer is always copied to location 1 of the Output Buffer. This may cause unexpected results, if unlike formats are present. You should be aware of the way that SELECTIT handles RDW fields (where applicable):

INPUTn Format	OUTPUT n Format	RDW Considerations
F / U	F / U	None (not applicable)
F / U	V	RDW will not generally contain valid data
V	F / U	RDW will be treated as data

V	V	RDW will be treated as RDW

Buffer Copies - INPUTn to UBUFFx
When copying records from an Input Buffer to a User Buffer, no reformatting will take place. If the Input Buffer contains a V format record, the RDW will be copied to the User Buffer, along with the data portion of the record.

Buffer Copies - UBUFFx to OBUFF
When copying records from a User Buffer to the Output Buffer, no reformatting will take place. If the Output Buffer is in V format, it is your responsibility to ensure that the first four (4) bytes of the User Buffer contain a valid RDW.

Output Files - Record Length

When you wish to write an output record from the Output Buffer, the length of such a record is normally determined by the 'data length' associated with the Output Buffer and the RECFM specified for the output file to which the record is to be written.

In the case of a V-format Output Buffer, the RDW field in the first four bytes of the Buffer is used to determine the length of the output record.

You may override this implied length by moving a value into 'outddname:DATALEN' prior to a 'SELECT TODD(outddname)', or by coding the LENGTH(mmm) operand on the SELECT statement. If the Output Buffer is in Variable format, then the record length in columns 1 and 2 will be used as the default for I/O to the file, unless overridden by DATALEN or LENGTH(mmm).

For V or U format records, if the length is available from more than one source, then the order of precedence is as follows:

1. The LENGTH(mmm) operand of SELECT
2. The outddn:DATALEN value (if non-zero)
3. The 'record length' of the record in the Output Buffer (for a V-format Output Buffer, the RDW will be used).

Output Buffer - Length of Record Written
When a record is written from the Output Buffer to an output file, both the format of the Output Buffer and the RECFM value for the output DDname ('outddn') are used to determine the actual record length, as follows:

OBUFF Format	outDDn RECFM	Start	Length of record written (Note 1)
F / U	F	1	outDDn:LRECL
F / U	U	1	Precedence order (Note 2)
F / U	V	1	Precedence order (Notes 2, 3) (RDW inserted by SELECTIT)
V	F	5	outDDn:LRECL
V	U	5	Precedence order (Notes 2, 4, 5)
V	V	1	Precedence order (Notes 2, 4) (RDW assumed in OBUFF:1/4 unless overridden)

Note 1 - Length Restrictions
When the length of the record being written exceeds the 'record length' associated with the Output Buffer, the PADCHAR character will be used to pad the record. The length is subject to a maximum value of outddn:LRECL (for F and V records) or outddn:BLKSIZE (for U records).

Note 2 - Precedence Order
The precedence order is listed under 'Output Files Record Length' above.

Note 3 - F/U copy to V
The record written will consist of the RDW plus the contents of OBUFF. If the 'record length' was used, the RDW will be set to a value 4 greater than the 'record length'.

Note 4 - RDW portion of OBUFF
If LENGTH(mmm) is omitted and outddn:DATALEN is zero, the first four bytes of the Output Buffer are assumed to contain a valid RDW. The value may not be identical to the 'record length' associated with the Output Buffer, especially if ALTER commands involving Expansion/Compression have taken place, or a record has been moved from a User Buffer into the Output Buffer.

Note 5 - V copy to U
If the RDW was used to determine the length of the output record, the actual length will be 4 less than the value in the RDW.

Shared Input / Output Buffer

To avoid unnecessary movement of data between internal buffers, SELECTIT will attempt to avoid using separate Input and Output Buffers whenever possible.

SELECTIT will use a separate Output Buffer when any one of the following conditions is detected in the command statements during the 'compile' phase:

1. Bytes 1 and/or 2 of a variable-length record (i.e. the RDW length field) are modified.
2. The LRECL of any output file referenced in a command statement is greater than INPUT1's LRECL.
3. Either an 'ALTER OBUFF COPYBUFF(buffer).' or a 'GETNEXT COPYBUFF(YES) ...' command is present.
4. An ALTER statement is present, where the source is a buffer field rather than a literal.
5. A 'floating' ALTER is present.
6. One or more of INPUT2 through INPUT9 is referenced by a command (the presence of these DDnames in the JCL does not constitute a reference).
7. A 'SELECT ... LENGTH(nnn).' command is encountered.
8. The target field of ALTER is specified with OBUFF:UCTRx.
9. 'OPTIONS OBFORCE(ON)' is in effect.

Occasionally, SELECTIT's choice of a single buffer will produce unexpected results; for example:

```
IF COL1 = C'A' THEN DO.
   ALTER COL1 COPY C'B'.
   SELECT TODD(OUTFILE).
END.
...
IF COL1 = C'A' THEN ...
```

The second IF will never find 'A' in column 1, since the previous action changed it to a 'B'. To overcome this type

of situation, you can force SELECTIT to use two buffers, by coding the following statement (see Rule 9 above) at the start of the SYSIN stream, or as a PARM:

```
OPTIONS OBFORCE(ON).
```

Note that it is not necessary to code an executable statement to force a separate Output Buffer (Rules 1 through 8 above).

Floating Field Handling

Fixed form record layouts are relatively easy to manipulate since we 'know' where every piece of data is (account number is a 6 byte field in location 4, status is a 1 byte flag at location 22, etc. etc.). Many times however, data must be manipulated which is 'free form', just as the text you are reading now is 'free form'.

Particular data items may have to be located by their relationships with other data around them; sometimes just a simple search or scan is needed; other times the scan may be combined with modification of the located data.

The 'floating' field support of SELECTIT has been provided to allow you to access and manipulate this type of data. The support allows you to scan buffers for the presence (or absence) of some data value, and optionally modify data based on the 'found' location within the buffer.

The 'scan' and 'modify' features operate in a manner similar to the 'find' and 'change' commands of an editor.

Just as a reminder, the format of a 'floating' operand is:

```
[prefix:]0[{+adj | -adj | +PLOC | +PLOCEND | +FLOCTERM}][/{length |
UCTRy | =}]
```

The full description of the operands will not be repeated here; review "Floating Buffer Fields" if needed.

Scanning for Strings

'Normal' scan requests Scanning for data strings within a buffer is performed by the IF statement. The use of a numeric '0' (zero) as the location within the buffer serves as a 'trigger' to SELECTIT to invoke the scanning function. e.g If an IF command is coded in the form

```
IF prefix:0 = operand/length ... THEN ...
```

or

```
IF operand/length = prefix:0 ... THEN ...
```

it will be taken as a request to search the buffer specified by 'prefix' (default INPUT1) for 'operand', beginning at the first byte of the buffer. 'operand' may be a:

- buffer field
- Keyword variable
- or a literal

'length' is the length of 'operand' (unless implied by the operand itself.

'operator' may only be one of:

- EQ (or '=')
- NE (or '¬=' or '<>')
- MASK (for partial matching of character patterns).

For EQ and MASK, the comparison will evaluate to true only if 'operand' is found within the buffer. This form may be interpreted as 'if the buffer contains 'operand' ...'

For NE, the comparison will evaluate to true only if 'operand' is not found within the record. This form may be interpreted as 'if the buffer does NOT contain 'operand' ...'

A 'floating' IF is identical in operation to the 'find' command of an editor.

Starting the scan at other than position 1 of the buffer The form '0+nnn' may be used instead of '0', to indicate that the search is to begin at location 'nnn+1' of the record. For example:

```
IF prefix:0+5 = operand/length ... THEN ...
```

or

```
IF operand/length = prefix:0+5 ... THEN ...
```

requests that the search begin at location 6 of the specified buffer.

Starting the scan based on a prior scan's result The form '0+PLOC' may be used instead of '0', to indicate that the search is to begin at the location following the leftmost location of the most recently executed floating IF test which gave a 'data found' result.

The form '0+PLOCEND' may be used to indicate that the search is to begin at the location following the rightmost location of the most recently executed floating IF test which gave a 'data found' result.

For example:

```
IF prefix:0+PLOC = operand/length ... THEN ...
IF operand/length = prefix:0+PLOCEND ... THEN ...
```

If no previous 'floating' IF has been executed, or all thosepreviously executed resulted in 'data not found', then '0 +PLOC' or '0+PLOCEND' will be treated as if '0' had been coded.

Processing considerations following the 'IF' When a 'floating' search condition results in 'data found', SELECTIT will 'remember' both the location and the length of the field, for use in subsequent 'floating' IF and ALTER commands; this holds true even across reads from input files, and other non-floating conditions.

Note: In the case of the 'NE' (or '¬=', or '<>') operator value on floating IF conditions, the 'data found' condition is the opposite of the 'true' condition.

The start location is stored in the Keyword variable FLOC, and the length is stored in Keyword variable FLEN. The location of the last byte of the 'floating' field is stored in Keyword variable FLOCEND.

You should be aware that only one location/length combination is 'remembered' by SELECTIT, so that if you test an Input Buffer, and subsequently modify the Output Buffer, you may not achieve your intended result. However, you may use the 'buffer:UCTRx' forms of buffer fields to overcome this difficulty.

Starting the scan based on a Delimited ALTER copy The form '0+FLOCTERM' may be used instead of '0', to indicate that the search is to begin at the location of the found delimiter in the most recently executed successful delimited ALTER statement. If no previous successful delimited ALTER has been executed, '0+FLOCTERM' will be treated as if '0' had been coded.

Changing Strings

General Description The general syntax of a floating ALTER is:

```
ALTER prefix:0+nnn¦-nnn COPY source.
```

After a 'floating' IF command has been executed which results in a 'data found' condition, subsequent ALTER ... COPY commands may use location '0', '0+nnn', or '0-nnn' in place of a normal fixed location. 0 represents the location of the leftmost byte of the most recently found string satisfying the condition:

```
IF prefix:0+xxxxx = operand
... THEN ...
```

0+nnn represents 'nnn' bytes to the right of the location represented by '0'.
0-nnn represents 'nnn' bytes to the left of the location represented by '0'.

Floating Field Examples

The following examples assume that the INPUT1 Buffer and the Output Buffer each contain the value:

```
   C'ABCDEFGH321IJK12345VWXYZ321'
loc: 00000000011111111122222222
     12345678901234567890123456 7
```

In the following examples, the portion of the record which has been modified, is shown with an underscore.

Example 1

```
IF 0 = C'321' THEN ALTER 0 COPY C'123'.
```

replaces the first occurrence of '321' with '123', with a resulting Output Buffer value of:

```
C'ABCDEFGH123IJK12345VWXYZ321'
```

FLOC will be set to 9, FLEN will be set to 3, and FLOCEND will be set to 11; locations 9-11 of the Output Buffer are modified.

Example 2

```
IF 0 = C'321' THEN ALTER LOC5 COPY 0+3/3.
```

replaces locations 5-7 with the 3 bytes following the first occurrence of '321', with the Output Buffer containing:

```
C'ABCDIJKH321IJK12345VWXYZ321'
```

As in the previous example, FLOC=9, FLEN=3, and FLOCEND=11; locations 5-7 of the Output Buffer are modified.

Example 3

```
IF 0+10 = C'321' THEN ALTER 0-3 COPY C'PQR'.
```

replaces the 3 bytes preceding '321' with 'PQR' (provided that '321' occurs in location 11 or greater). The Output Buffer will contain:

```
C'ABCDEFGH321IJK12345VWPQR321'
```

In this case, FLOC=25, FLEN=3, FLOCEND=27; and locations 22-24 of the Output Buffer are modified.

Example 4 - Compression and Expansion

After 'ALTER 0 COPY source' is executed, 'source' will replace the field specified in the most recently executed true 'floating' comparison, with expansion or compression if necessary. This is identical in operation to the 'change' command of an editor.

In addition, the Keyword variable FLCHG will contain the signed number of bytes compressed or expanded; the value will be negative if compression was performed, positive if expansion was performed, and zero if no length change was detected.

Note: Keyword variable DATALEN is not changed as a result of compression or expansion. If compression is performed, the output record will be padded on the right with the OPTIONS PADCHAR value. For example:

```
IF 0 = C'12345' THEN ALTER 0 COPY C'321'.
```

will replace the original five-byte field '12345' with the three-byte field '321', shift the remainder of the record two bytes to the left, and append two pad characters to the end of the output record.

Using the same original values of INPUT1 and OBUFF and a pad character of blank, the Output Buffer will contain:

```
C'ABCDEFGH321IJK321VWXYZ321 '
```

In this case, FLOC=15, FLEN=5, FLOCEND=19, and FLCHG will be set to -2 (indicating a 2-byte compression); locations 15-27 of the Output Buffer are modified.

If expansion is performed, and the resulting output record would be longer than the buffer, then truncation will occur on the right. You can prevent compression or expansion from taking place by the use of the special length value '/=' on the first operand, as follows:

```
IF 0 = C'12345' THEN ALTER 0/= COPY C'321'.
```

This will replace only the first 3 bytes of the original five byte field '12345' with the three-byte field '321'; the remaining locations in the record are not affected. Using the same original values of INPUT1 and OBUFF, the Output Buffer will contain:

```
C'ABCDEFGH321IJK32145VWXYZ321'
```

In this case, FLOC=15, FLEN=5, FLOCEND=19, and FLCHG will be set to 0 (indicating no compression); locations 15-17 of the Output Buffer are modified.

Variable Length Records If a variable length record is either compressed or expanded, SELECTIT will not alter the length field in the output record's RDW. If you wish to maintain the RDW value, you will need to add the following after the 'ALTER 0 ...' statement:

```
ALTER buffer:1/2 ADD FLCHG.
```

Alternatively, you could use the Keyword variable DATALEN, or the LENGTH operand of 'SELECT TODD' for output records.

WARNING You should only use 'floating' locations on ALTER, if that ALTER command is subordinate to the corresponding 'floating' IF (see examples below). SELECTIT will retain information from a true 'floating' condition, even across multiple input records; a false 'floating' condition does not cause a previous true condition to be 'forgotten'.

If no previous true 'floating' IF has been executed at the time that a 'floating' ALTER is executed, then 'ALTER 1 ...' will be assumed, with the length value(s) determined as for normal fixed ALTERs. Note that if a 'floating' IF test is part of a multiple condition which evaluates to false, but the individual floating IF test is successful, the appropriate values from that 'true' test are still retained for subsequent use.

Example 5
Search the primary input record for the string 'ANYWHERE' and change it to the string 'SOMEWHERE':

```
IF 0 = C'ANYWHERE'
   THEN ALTER 0 COPY C'SOMEWHERE'.
```

Note that the output record will be expanded by 1 byte (with FLCHG set to +1).

Example 6
Search the INPUT2 record for the string 'GOODBYE' in location 20 or later, and change it to the string 'HELLO':

```
IF INPUT2:0+19 = C'GOODBYE' THEN DO.
   ALTER OBUFF COPYBUFF(INPUT2).
   *: Move to Output buffer.
   ALTER 0 COPY C'HELLO'.
   END.
```

Note that the output record will be compressed by 2 bytes (with FLCHG set to -2).

Example 7
Search the INPUT3 record for the string 'ANYWHERE', then search for the string 'ELSE' following it; if found, change 'ELSE' to spaces.

```
IF INPUT3:0 = C'ANYWHERE' THEN DO.
   IF INPUT3:0+PLOCEND = C'ELSE' THEN DO.
      ALTER OBUFF COPYBUFF(INPUT3).
      ALTER 0 COPY 4C' '.
   END.
```

```
END.
```

In this example, suppose that 'ANYWHERE' is found in locations 10-17 of INPUT3, then the string 'ELSE' in columns 17-20 would satisfy a '0+PLOC' condition, even though this might not have been the intention. The '0 +PLOCEND' will cause the search for ELSE' to start in column 18.

Predefined Keyword Variables

Keyword variables provide you with access to data other than the data within your actual data records. Most of the variables are 'read only', that is they may not be used as the 'target' of ALTER commands. The details provided in the following sections describe:

- The Keywords available
- What data is accessed by the keyword
- Valid usages of the keyword
- Whether or not you may modify the keyword data directly

The charts which follow use abbreviations to describe certain usage characteristics, the explanation for those abbreviations is provided here.

Following the charts are detailed descriptions of each keyword variable.

Pfx - Indicates the type of prefix (if any) allowed
no = no prefix allowed on this field
bb = field is prefixable by a buffer name (INPUT1-9, OBUFF, or UBUFFA-Z)
dd = field is prefixable by a ddname (INPUT1-9 or outputddname)
ddi = field is prefixable by an input ddname (INPUT1-9)
ddo = field is prefixable by an output ddname

Adj. - Indicates whether +/- adjustments are allowed
no = no +/- adjustment allowed
+/- = may have a +nnn or -nnn adjustment, or +xxx Keyword variable
+ = may have a +nnn adjustment

Type - Indicates type of field
R/O = read-only
U/M = user-modifiable

I/O Related Fields and Variables

(Default prefix INPUT1:)

Pfx	Name	Adj	Type	Fmt	Implicit Length	Contents
dd	CREC	no	R/O	F	4	Record Number (last reread/write)
dd	DATALEN	no	R/O	F	4	length of last record read
ddo	DATALEN	no	U/M	F	4	length of next record to be written
dd	DCB	+	R/O	X	4 *** 96 104	DCB (Sequential DCB length) (Keyed DCB length)
dd	DSNAME	+	R/O	C	44	Dataset name
dd	DSNAMEX	+	R/O	C	54	Dataset name including membername

dd	DSORG	no	R/O	X	1	DSORG (in DB)
no	IMEMBER	no	R/O	C	8	Input membername during PDS mode processing
dd	JFCB	+	R/O	X	176	JFCB
dd	LRECL	no	R/O	H	2	LRECL value (in DCB)
no	OMEMBER	no	R/W	C	8	Output membername during PDS mode processing
dd	NVOLS	no	R/O	X	1	No. of volumes (in JFCB)
dd	VOLSER	+	R/O	C	6 *** (30)	First volser for DSNAME (second volser at VOLSER+6, etc. Max 5

*** = field is longer than its implicit length

Date/Time related Fields and Variables

Pfx	Name	Adj	Type	Fmt	Implicit Length	Contents
no	DATETIME	+	R/O	X	8 *** (39)	date/time (see below) actual length
no	DAY	no	R/O	C	2	day of month (e.g. C'03')
no	JULIAN	no	R/O	C	5	Julian date (yyddd)
no	MONTH	no	R/O	C	3	Month name (e.g. C'SEP')
no	MONTHNUM	no	R/O	C	2	Month number (e.g. C'02' for February)
no	TIME	no	R/O	C	5	Time (hh:mm)
no	YEAR	no	R/O	C	2	Year (e.g. C'12')
no	CENTURY	no	R/O	C	2	Century (e.g. C'20')
no	FULLYEAR	no	R/O	C	4	Full year (e.g. C'2012')

*** = field is longer than its Implicit Length

Format of DATETIME field

```
(8 bytes)   X'yydddhhmmsstttt' (julian date/time in hex)
(39 bytes)  C'ddmmmyy (yyddd) hh:mm:ss.ttttt yyyymmdd'
              ¦         ¦       ¦               ¦
              ¦         ¦       ¦               ¦
             (a)       (b)     (c)             (d)
              ¦         ¦       ¦               ¦
DATETIME+8 ---+         ¦       ¦               ¦
DATETIME+17 -----------+        ¦               ¦
DATETIME+24 ------------------+                 ¦
DATETIME+39 ------------------------------------+
```

(a) is used for DAY/MONTH/YEAR (dd/mmm/yy)

(b) is used for JULIAN (yyddd)

(c) is used for TIME (hh:mm)

(d) is used for CENTURY/FULLYEAR/MONTHNUM

Miscellaneous Fields

Pfx	Name	Adj	Type	Fmt	Implicit Length	Contents
no	HEADING	no	U/M	C	54	TITLE value
no	JOBNAME	no	R/O	C	8	Jobname
no	LINECTR	no	U/M	P	2	Line counter (SYSPRINT)
no	PROCNAME	no	R/O	C	8	Name of PROCSTEP
no	RCODE	no	U/M	F	4	Return code (init F'0')
no	STEPNAME	no	R/O	C	8	STEPNAME
no	UCTRA . . UCTRZ	no	U/M	F	4	User Counters (iniy F'0')
no	USERPARM	no	R/O	C	8	Data from JCL PARM field

'floating' Field Variables

IF/ALTER 0+xxxx fields

Name	Significance
0+FLOCTERM	delimiter location from last delimited ALTER/COPY
0+PLOC	last floating IF location + 1
0+PLOCEND	last floating IF location + length

IF/ALTER result fields (no prefix allowed)

Pfx	Name	Adj	Type	Fmt	Implicit Length	Contents
no	FLCHG	no	R/O	F	4	sign bytes compressed or expanded on last non-fixed ALTER
no	FLEN	no	R/O	F	4	length of last ALTER source field
no	FLOC	no	R/O	F	4	last floating IF location
no	FLOCEND	no	R/O	F	4	last floating IF location + length - 1

Alphabetical Descriptions

CREC CREC is a fullword binary counter (implicit length 4) containing the record number of the last record read from an input file, or of the last record written to an output file. CREC may not be modified by any of your input commands. CREC may only be prefixed by a DDname; if no

prefix is used, INPUT1 will be assumed.

For INPUT1-9 DDnames, the appropriate INPUTn:CREC is incremented by 1, every time that a GETNEXT is executed against DDname INPUTn (including the implicit GETNEXT against INPUT1 which occurs at the internal label RETURN).

Note: For end-of-file on INPUT1-9, CREC will not be incremented, and will contain the number of the last record read from the file. If INPUTn references a concatenation, INPUTn:CREC will contain the cumulative total of records read for DDname INPUTn.

For output DDnames, the appropriate ddname:CREC is incremented by 1 every time that a SELECT TODD(ddname) is executed.

All CREC values are initialized to zero (0) by SELECTIT at 'compile' time.

DATALEN DATALEN DATALEN is a fullword binary counter (implicit length 4) containing the length of the last record read from an input file,

or the length of the next record to be written to an output file. DATALEN may be modified by your commands for output DDnames only; for INPUTn DDnames, DATALEN may not be modified. DATALEN may only be prefixed by a DDname; if no prefix is used,

INPUT1 will be assumed.

Note: DATALEN should not be confused with LRECL, which contains the DCB LRECL value.

For INPUT1-9 DDnames, the appropriate INPUTn:DATALEN is set to the length of the last record read when a GETNEXT is executed against DDname INPUTn.

For output DDnames, a zero value in ddname:DATALEN indicates that SELECTIT is to use the LENGTH operand of SELECT (or the RDW field in the Output Buffer). A non-zero value indicates the length of the next RECFM U or V output record to be written (unless overridden by the LENGTH operand of SELECT); a non-zero value will be ignored for RECFM F output. The appropriate ddname:DATALEN is reset to zero, after a SELECT TODD(ddname) is executed.

All output file DATALEN values are initialized to zero (0) by SELECTIT at 'compile' time.

DATETIME DATETIME is the name of the read-only field containing the date and time of the start of the current SELECTIT run. Its implicit length is 8; its total length is 47. It may not be prefixed. It may be adjusted by a '+nnn' value.

The format of DATETIME field is as follows:

```
(8 bytes)   X'yydddhhmmsttttt' (julian date/time in
hex)
(39 bytes)  C'ddmmmyy (yyddd) hh:mm:ss.ttttt yyyymmdd'
```

See also Keyword variables DAY, JULIAN, MONTH, TIME, and YEAR.

DAY DAY is a 2-byte read-only field containing the numeric day of the month in character format. It may not be prefixed.

Refer to Keyword variables DATETIME for additional details.

DCB DCB is the name of the read-only field which contains a DCB. Its implicit length is 4 bytes; the actual length of the field depends on the organization of the data set (currently 96 bytes for sequential data sets, 104 bytes for a keyed data set). It may only be prefixed by a DDname; if no prefix is used, INPUT1 will be assumed. It may be adjusted by a '+nnn' value.

Refer to the IBM publication 'DATA AREAS' for a full description of the fields within a DCB.

See also Keyword variables LRECL, DSORG.

DSNAME DSNAME is a 44-byte read-only field containing the name of the data set. It may be prefixed by INPUT1-9 or an output DDname; if no prefix is used, INPUT1 will be assumed. It may be adjusted by a +nnn value.

For INPUT1-9, if the ddname references a concatenation, DSNAME will contain the name of the data set currently being read.

DSNAME is part of the Keyword variable JFCB. See also DSNAMEX.

DSNAMEX DSNAMEX is a 54-byte read-only field containing the name of the current data set. It is identical to the DSNAME field in all respects, except that if the current dataset is a member of a PDS, DSNAMEX will contain a value of the form 'dsname(member)'.

DSORG DSORG is a 1-byte binary read-only field containing the code for the data set organization. It may only be prefixed by a DDname; if no prefix is used, INPUT1 will be assumed.

Refer to Keyword variable DCB for additional details.

FLCHG FLCHG is a fullword binary read-only field (implicit length 4) containing the signed number of bytes compressed or expanded during the previous non-fixed ALTER. If the record was expanded, the value will be positive; if the record was compressed, the value will be negative.

FLEN FLEN is a fullword binary read-only field (implicit length 4) containing the length value used during the previous floating IF or delimited ALTER ... COPY.

See also FLOCTERM.

FLOC FLOC is a fullword binary read-only field (implicit length 4) containing the location of the start of the found string in the previous floating IF statement.

See also FLOCEND, PLOC, PLOCEND.

FLOCEND FLOCEND is a fullword binary read-only field (implicit length 4) containing the location of the end of the found string in the previous floating IF statement.

See also FLOC, FLOCEND, PLOCEND.

FLOCTERM FLOCTERM may only be used in the form '0+FLOCTERM' as the target field of an ALTER command. It describes the location of

the delimiter in the previous delimited ALTER ... COPY statement.

HEADING HEADING is a 54-byte user-modifiable field containing the current TITLE value. It may not be prefixed. Its value may also be altered by coding a TITLE statement.

If HEADING is modified, the source field must be 54 bytes in length. If shorter, previous contents of the field may still be visible.

INPUT1-9 INPUT1 through INPUT9 are the names of the Input Buffers. INPUTn may be used as the buffer name in an ALTER/COPYBUFF command, in the FILE(INPUTn) parameter of GETNEXT, or as a prefix for a field on IF and ALTER statements.

There must be a DD statement present for each INPUTn referenced in a SYSIN command.

JFCB JFCB is the name of the read-only Keyword variable which contains the JFCB. Its implicit length is 176. It may only be prefixed by a DDname; if no prefix is used, INPUT1 will be assumed. It may be adjusted by a '+nnn' value.

Refer to the IBM publication 'DATA AREAS' for a complete description of the JFCB.

This field includes the Keyword variables DSNAME, NVOLS, and VOLSER. See also Keyword variable DSNAMEX.

JOBNAME JOBNAME is an 8-byte read-only field containing the name of the currently executing job. It may not be prefixed.

JULIAN JULIAN is a 5-digit read-only field containing the julian date of the current run date in character format. It may not be prefixed.

Refer to Keyword variable DATETIME for additional details.

LINECTR LINECTR is a 2-byte packed field used by SELECTIT to determine when a page-break should occur on SYSPRINT. LINECTR may be modified by SYSIN commands. LINECTR may not be prefixed.

If LINECTR is modified, the source must be a 2-byte packed field.

You may set LINECTR to PL2'0' if you wish to force a page-break.

LRECL LRECL is a halfword binary read-only field (implicit length 2) containing the DCB LRECL value for a file. It may only be prefixed by a DDname; if no prefix is used, INPUT1 will be assumed.

Refer to Keyword variable DCB for additional details.

Note: LRECL should not be confused with DATALEN, which is used to indicate the 'record length' of the current input or output record.

MONTH MONTH is a 3-byte read-only field containing the abbreviated month name (e.g. C'MAR') of the current run date. It may not be prefixed.

Refer to Keyword variable DATETIME for additional details.

NVOLS NVOLS is a 1-byte binary read-only field containing the number of volumes for the data set. It may only be prefixed by a DDname; if no prefix is used, INPUT1 will be assumed. The volume serial number(s) for the data set are available in the Keyword variable VOLSER. However, if NVOLS is greater than 5, only the first 5 volume serial numbers are available.

NVOLS is part of Keyword variable JFCB. See also Keyword variable VOLSER for additional details.

OBUFF OBUFF is the name of the Output Buffer. OBUFF may be used as the buffer name in an ALTER/COPYBUFF command, or as a prefix for a field on IF and ALTER statements.

outddn 'outddn' may only be used as a prefix for the I/O related Keyword variables. There must be a DD statement present for each such prefix in the SYSIN commands.

PLOC PLOC may only be used in the form '0+PLOC' on an IF statement, to indicate that a floating search is to start at the byte following the starting location of the previous true floating IF.

See also FLOC, FLOCEND, PLOCEND.

PLOCEND PLOCEND may only be used in the form '0+PLOCEND' on an IF statement to indicate that a floating search is to start at the byte following the ending location of the previous true floating IF.

See also FLOC, FLOCEND, PLOC.

PROCNAME PROCNAME is an 8-byte read-only field containing the procstepname of the current job. It may not be prefixed.

RCODE RCODE is a fullword binary counter (implicit length 4) containing the return-code with which SELECTIT will terminate. It is modifiable by SYSIN statements, but no prefix may be specified. RCODE is initialized to zero (0) by SELECTIT at 'compile' time.

When SELECTIT terminates, the value in RCODE will be used as the return-code; for this reason, any value placed in RCODE should be in the range 0-4095. Any value outside this range will cause unpredictable results (refer to the COND parameter of the EXEC statement in the IBM JCL manuals for details).

STEPNAME STEPNAME is an 8-byte read-only field containing the stepname of the current job. It may not be prefixed.

TIME TIME is a 5-byte read-only field containing the time at the start of the current run, in the format C'hh:mm'. It may not be prefixed.

Refer to Keyword variable DATETIME for additional details.

UBUFFA-Z There are up to 26 user buffers (UBUFFA through UBUFFZ), each with a length equal to the

OPTIONS UBSIZE setting (default value 1024). UBUFFx may be used as the buffer name in an ALTER/COPYBUFF command, or as a prefix for a field within a user buffer on IF and ALTER commands.

SELECTIT will create only those buffers referenced by SYSIN commands; the contents of the buffers are unpredictable until you move data into them with an ALTER command.

UCTRA-Z There are 26 user counters (UCTRA through UCTRZ), each a binary fullword with an implied length of 4. SELECTIT will initialize all 26 UCTRx fields to F'0' at 'compile' time. You may modify the value of UCTRx with SYSIN commands. SELECTIT will print all non-zero UCTRx values at the end of the run (provided that SYSPRINT is present).

If any UCTRx value is modified, the source field must be 4 bytes in length.

UCTRx has 3 uses:

1. A counter value (without a prefix). The value of the UCTRx field is considered to be just a numeric value.
2. A buffer field location (which must be preceded by a buffer prefix). The UCTRx field is considered to contain a location within a buffer.
3. A field length specification (following the '/' which indicates a length value). The UCTRx field is considered to contain a length value.

SELECTIT differentiates between cases 1 and 2 above, solely based on the absence or presence of the buffer prefix. If UCTRx is not prefixed, it is assumed to represent a number; if it is prefixed, it is assumed to represent a location. When used as a buffer field location (case 2 above), the following limitations should be noted:

- In order to distinguish between the value of the UCTRx field, and its use as a location, a buffer prefix is required to identify the field as a location value.
- Zero values, negative values, and values in excess of the buffer length will cause S0C4 abends, or other unpredictable results. SELECTIT will not validate the contents of the UCTRx field, when it is used as a location.
- A value of zero (0) will not be interpreted as a 'floating' request (see also the previous item).

When used as a buffer field length (case 3 above), the following limitations should be noted:

- Zero values, negative values, and values in excess of the buffer length will cause S0C4 abends, or other unpredictable results. SELECTIT will not validate the contents of the UCTRx field, when it is used as a length value.
- When used as a length field on the first operand of an IF statement, and implied subjects are in use, the value will not be 'remembered' on the second and subsequent comparisons:

  ```
  IF subj/UCTRA = loc1 OR loc2 ...
  ```

  is invalid; the correct construction is:

  ```
  IF subj = loc1/UCTRA OR loc2/UCTRA ...
  ```

VOLSER VOLSER is the name of the read-only Keyword variable containing the volume serial number(s) of the current data set. Its implicit length is 6; the total length is 30 (enough for 5 volume serial numbers); the Keyword variable NVOLS can be used to determine the actual number of volumes. It may only be prefixed by a DDname; if no prefix is used, INPUT1 will be assumed. It may be adjusted by a '+nnn' value.

If NVOLS is greater than 5, then only the first 5 volume serial numbers are available. If more than one volume serial is present, you may access them as follows:

- 1st volume - 'prefix:VOLSER'
- 2nd volume - 'prefix:VOLSER+6'
- 3rd volume - 'prefix:VOLSER+12'
- 4th volume - 'prefix:VOLSER+18'
- 5th volume - 'prefix:VOLSER+24'

VOLSER is part of the Keyword variable JFCB. See also Keyword variable NVOLS.

YEAR YEAR is a 2-digit read-only field containing the numeric year of the current run date in character format. It may not be prefixed.

Refer to Keyword variable DATETIME for additional details.

Performance Considerations

If you are not familiar with the items covered in "Basic Performance Considerations" you should be sure to review that section as well as the points covered here.

Buffer Handling

Variable Record Processing
If you are processing a RECFM=V input file which has both a large LRECL and which contains widely different record lengths then the point in "Shared Input/Output Buffer" should be reviewed carefully. If your logic is such that SELECTIT cannot share the Input and Output buffers you may incur high overheads since each input record will be moved to the output buffer immediately following the read. Since SELECTIT copies and pads the record up to the maximum size with the OPTIONS PADCHAR value. If, for example, the characteristics of your file were

```
RECFM=VB,LRECL=32000,BLKSIZE=32004
```

but the average record length was actually only 500 bytes, SELECTIT would be forced to do a buffer fill of 31500 bytes for every record read.

If the basic operation of the step is to search and select only certain records to the output file then considerable savings can be achieved by coding the operation as follows:

```
OPTIONS AUTOCOPY(OFF).
IF selection-criteria THEN DO.
   ALTER OBUFF COPYBUFF(INPUT1).
   SELECT TODD(OUTPUT1).
END.
```

The AUTOCOPY(OFF) request tells SELECTIT to bypass the automatic copying of the input record to the output buffer. This avoids the high overhead moving and padding of each record before you have even determined that the record is needed. However, this means that you must code your own ALTER OBUFF COPYBUFF(INPUT1) statement when your logic has determined that this record is to be selected to the output file.

User Buffer size
You should review your use of the UBUFFx areas and set the OPTIONS UBSIZE(...) to the lowest value that meets your needs. Any values greater than this minimum are simply wasteful.

EOJ Determination

"How end-of-job is determined" reviews the methods by which SELECTIT determines when to terminate. Because the default for a SELECT statement is RECS(ALL), it is all too easy to create circumstances which force SELECTIT's determination of EOJ to be incorrect and thereby waste resources.

For example take a transaction file which contains a header record at its beginning (identified by 'C01' in byte 1) followed by detail records (identified by 'T' in byte 1). A user wished to dump the header record and the first 10 detail records and set up the following statements:

```
IF BYTE1 = C'C01' SELECT PRINT(DUMP).
IF BYTE1 = C'T' SELECT PRINT(DUMP) RECS(10).
```

With this setup, SELECTIT will be forced to read the entire input file since it has no way of 'knowing' that only 1 'C01' record exists. By simply adding RECS(1) to the SELECT for the 'C01' record SELECTIT will terminate immediately following the eleventh record since the RECS(...) count for all SELECT statements will have been satisfied.

Keyed File Considerations

The items described in this chapter should only be considered when your access to the file is dependent on it having keys. i.e. you are using the SKEY and/or EKEY operands of the SELECT command. If you are processing the keyed file sequentially these considerations do not apply; treat the file as you would any other sequential file.

Using SKEY/EKEY

The SKEY/EKEY operands of the SELECT command allow you to restrict the actions of the particular SELECT to a specific range of keys within the keyed input file. **Note**: When using SKEY/EKEY the keyed file must be allocated to the INPUT1 DD, keyed support is not available on the INPUT2-INPUT9 files.

For example, to copy only records with keys between C'FFF' and C'KKK' inclusive to a new file, the SELECTIT statement would be:

```
SELECT SKEY(C'FFF') EKEY(C'KKK') TODD(OUTPUT1).
```

Multiple ranges can also be handled.

```
SELECT SKEY(C'B01') EKEY(C'B99') TODD(OUTPUT1).
SELECT SKEY(C'R01') EKEY(C'R99') TODD(OUTPUT1).
SELECT SKEY(C'X01') EKEY(C'X50') TODD(OUTPUT1).
```

The above example would select all records with keys from B01 thru B99, R01 thru R99, and X01 thru X50.

When SKEY/EKEY are present in the command stream, SELECTIT will determine what ranges are to be processed and will cause only those records which fall into one of the requested ranges to be read. Fragmented and overlapping ranges may be specified on different SELECT statements; SELECTIT will 'collapse' and optimize the ranges so that no records need be read more than once.

The lowest SKEY value encountered on all SELECT statements will be the first keyed record read from the INPUT1 file; similarly, the highest EKEY value will determine the last record read.

SKEY/EKEY and CREC

When multiple ranges of keys are present on SELECT statements, SELECTIT will process records from only those ranges. This means that, depending on the ranges specified, some records of the dataset may not be read, even though they have keys greater then the lowest SKEY and lower than the highest EKEY.

Example (using 1-byte keys):

There are 2 SELECT statements, where one has `'SKEY(C'B') EKEY(C'D')'` and the other has `'SKEY(C'M') EKEY(C'Q')'`; INPUT1 consists of 26 records with key values C'A', C'B', C'C', ..., C'Z'.

The only records read from INPUT1 will be the 3 records with keys 'B' to 'D' and the 5 records with keys 'M' to 'Q'; the remaining 18 records will not be read, and will therefore never be processed by the SELECT statements.

For this reason, RECS, SREC and BY should be used with caution when processing keyed files. In the example

above, the record with key 'C' will be the second record processed by SELECT, even though it is the third record in the input file.

Reality: The above description is perfectly correct with regard to how the I/O performed relates to your control statements. In reality, SELECTIT actually must process 1 record past the end of each range. Careful examination of external I/O counts would confirm this. The logical description above is however the one that matters.

Processing PDS Datasets

This chapter describes the SELECTIT support for processing entire partitioned datasets. If you need to process a single member of a PDS, simply point to it directly in the JCL. e.g. **`//INPUT1 DD DSN=MY.PDS(MEMBER1),DISP=SHR`** and treat it as any other sequential dataset.

Activating PDS Support

To 'activate' SELECTIT's extended PDS support, you must add a PARM field to the EXEC statement which invokes SELECTIT as follows:

```
//STEP1 EXEC PGM=SELECTIT,PARM='PDSMODE(ON)'
```

If other PARM field operands are present, just add this operand.

```
//STEP1 EXEC PGM=SELECTIT,PARM='CAPS(OFF) PDSMODE(ON)'
```

Activating PDSMODE simply allows it to be requested for individual files. To activate for a particular file simply code the DSN parameter to reference the dataset without a member name. For input, you may only activate PDS processing for the INPUT1 DD; it may not be specified for the alternate inputs (INPUT2 - INPUT9). It may be specified for any desired output dataset.

PDS Processing

Once activated, PDS processing will cause SELECTIT to process your input commands once for each member of the INPUT1 dataset. When all members have been processed, the SELECTIT step will terminate. i.e. it is as if a separate SELECTIT step were executed for each member.

Each DDNAME for which PDS processing has been activated will be logically opened and closed as if the member name had been coded on the JCL statement(s). All other DDNAMEs will be handled normally. i.e. opened once per step, not once per member.

Membername Handling

The current member name from INPUT1 being processed is available in the keyword variable IMEMBER. At the start of each member's processing, this name is also copied to OMEMBER (see below).

If not altered, any output file which has been activated for PDS support will create the member name contained in OMEMBER. If you wish to control the member name(s) being created you should modify OMEMBER as needed before the first record is written to the dataset.

ALIAS Names

Any member names in the INPUT1 PDS which are ALIAS names will NOT be processed by SELECTIT, they are ignored.

PDF Statistics

Note that if updating PDS members and the original members were maintained under PDF (and thus had PDF member statistics) that these statistics will be lost during the update.

I/O Statistics

SELECTIT will, by default, only provide I/O statistics for the entire PDS, not at an individual member level. If member level statistics are desired, then include an OPTIONS MEMBSTAT(YES). as the first control statement.

Some examples of PDS usage can be found in:

"PDS Input Processing"

"PDS Update Processing".

Customizing the Runtime Environment

SELECTIT makes certain assumptions and takes certain defaults when it processes your data. The ability is provided to allow you to alter the normal defaults and request alternative processing.

Specifying the OPTIONS

Two methods are provided for you to specify what alternative options you wish:

EXEC statement PARM field Any of the SELECTIT OPTIONS parameters may be coded in the PARM of the EXEC statement which invokes SELECTIT. If coded as a PARM value, the word 'OPTIONS' and the terminating period may be omitted; the effective OPTIONS statement will be printed on SYSPRINT with statement number zero (00000). The PARM field may also used to pass user data to the SELECTIT program. See "User PARM Data" for more details.

OPTIONS control statement An OPTIONS statement may be the first statement in the SYSIN file. Statement syntax is covered by "OPTIONS".

Both the above may be used in the same run if desired. In the case of a parameter being specified in both places, they will be logically processed as if the PARM specified values precede the OPTIONS statement values. i.e. the OPTIONS specified value will be the one used.

OPTIONS Available

The options described below may all be invoked by either of the methods described above (i.e. PARM values, or an actual OPTIONS statement). Any of the following parameters may be entered in any order.

In all cases, the default setting is shown first; for all parameters which have a choice of 'ON' or 'OFF', 'YES' may be used interchangeably with 'ON', and 'NO' may be used interchangeably with 'OFF'.

Parameter	Significance
AUTOCOPY(ON | OFF)	INPUT1 copy control
CAPS(ON | OFF)	upper/lower case control
CBLKSIZE(ON | OFF)	Copy BLKSIZE to Output
DEFER(ddname)	defer OPEN of a DD name
EOFFILL(X'FF' | 1-byte-literal)	fill-character at EOF
EOFLABEL(EOJ | label)	GOTO label at EOF
LIST(ON | OFF)	SYSIN listing control
MEMBSTAT(OFF | ON)	Member level statistics
OBFORCE(OFF | ON)	separate Output buffer
OPCHECK(ON | OFF)	I/O check
PADCHAR(X'40' | 1-byte-literal)	pad character
PDSMODE(OFF | ON)	PDS Mode processing
RETLABEL(RETURN | label)	GOTO label at end of SYSIN
STATS(ON | OFF)	statistics print control
TRACENUM(ON | OFF)	trace overhead control

UBSIZE(1024 | numeric)	user buffer size

OPTIONS Command Placement If coded as part of the SYSIN stream, the OPTIONS command must precede any executable statements. Comments and TITLE statements may precede OPTIONS.

Placement (CAPS and LIST Parameters) The keywords may be entered in any order, but if you wish to use CAPS(OFF), it must be coded as a PARM value, or on the first physical card in SYSIN, in order for the titles on the first page of SYSPRINT output to appear in lower case. If CAPS(OFF) is encountered on the second (or subsequent) physical SYSIN statement, any lower case characters following CAPS(OFF) will be left as is, and titles on page 2 (and onward) of SYSPRINT will include lower case characters. See also the discussion on the CAPS parameter below for JCL requirements.

If you wish to use LIST(OFF), it too must be coded as a PARM value, or on the first physical card in SYSIN, in order to suppress the control card listing. If LIST(OFF) is encountered on the second (or subsequent) physical SYSIN statement, the page headings and all previous SYSIN cards will be listed on SYSPRINT.

Parameter	Default	Description
AUTOCOPY(ON | OFF)	ON	'ON' indicates that INPUT1 records are to be copied to the Output buffer on every 'GETNEXT FILE(INPUT1)'. This provides a default of COPYBUFF(YES) for GETNEXT FILE(INPUT1). 'OFF' indicates that the copy should not take place. This provides a default of COPYBUFF(NO) for GETNEXT FILE(INPUT1). AUTOCOPY has no effect on Input files INPUT2 through INPUT9. Note: There is an implicit GETNEXT at label RETURN. Any 'GETNEXT FILE(INPUT1)' in the SYSIN stream may override the AUTOCOPY setting by coding the COPYBUFF(YES¦NO) parameter on the GETNEXT command.
CAPS(ON | OFF)	ON	'ON' indicates that all SYSPRINT output (including 'SELECT PRINT(xxx)') should be converted to upper case. 'OFF' indicates that lower case characters should be left as is on SYSPRINT. Note: If you use CAPS(OFF), it is your responsibility to ensure that you have valid CHARS, FCB, etc., information on the SYSPRINT DD statement, and that you will be printing the output on a printer which supports lower case characters. See also the restriction above concerning placement of CAPS(OFF).
CBLKSIZE(ON | OFF)	ON	'ON' indicates that SELECTIT should provide a default BLKSIZE for output datasets by copying the value from the INPUT1 dataset. 'OFF' means that SELECTIT will leave the BLKSIZE as zero if none was

		provided in the JCL for the output DD. The latest versions of the operating system now provide an optimum BLKSIZE, based on the output device, for new output datasets if none is provided. It may therefore be advantageous to prevent SELECTIT from providing a default since this bypasses this automatic selection of the optimum BLKSIZE by the operating system.
DEFER(ddname)	None	The DEFER operand provides you with additional control over the timing of the OPEN performed against a particular DD name. Without using this parameter, SELECTIT OPENs all DD's during program initialization. This creates several problems, particularly when the input/output datasets are on tape: ○ Enough tape units must be allocated at one time to satisfy all the DD's in the JCL. WITH this parameter: ○ The JCL parameter UNIT=AFF=... can be used to reduce this simultaneous allocation. ○ Multiple output DD's can be used in one step to create multiple files on one tape. The DEFER operand, in conjunction with the CLOSE command, allows you to control the OPENing and CLOSEing of input and output files to 'work around' this problem. When you specify DEFER for a DD, SELECTIT will not OPEN the file until absolutely necessary. That is, for input files, not until a GETNEXT is issued for it; for output files, not until a SELECT TODD(...) is issued. For an example of using OPTION DEFER and CLOSE, see "Controlling OPEN/CLOSE of files".
EOFFILL(X'FF' | 1-byte-literal)	X'FF'	This parameter defines the character which will be used to fill an Input buffer when an end-of-file condition is detected on a GETNEXT command. The EOFFILL parameter is applicable to all Input files (including INPUT1).
EOFLABEL(EOJ | label)	EOJ	This parameter defines the default label to which control will be passed when an end-of-file condition is detected on a GETNEXT command with the EOF parameter omitted. The EOFLABEL parameter is applicable to all Input files (including INPUT1).
LIST(ON | OFF)	ON	'ON' indicates that all SYSIN statements are to be listed on SYSPRINT. 'OFF' indicates that no SYSIN statements are to be listed.

		If SYSPRINT is absent, LIST(OFF) will be assumed. Note: Non-zero UCTRx values will always be printed, even with LIST(OFF) and STATS(OFF), provided that SYSPRINT is present. See also the restriction above concerning placement of LIST(OFF).
MEMBSTAT(ON | OFF)	OFF	If SELECTIT is processing an entire PDS (see "Processing PDS Datasets".) this option controls whether I/O statistics should be provided for each member of the processed datasets. The default of OFF will provide only a single set of statistics for the entire PDS. ON will provide I/O counts for each member
OBFORCE(OFF | ON)	OFF	'OFF' indicates that SELECTIT is to follow its normal rules for determining whether a separate Output Buffer is required. 'ON' indicates that a separate Output buffer is to be created (unconditionally).
OPCHECK(ON | OFF)	ON	'ON' indicates that SELECTIT is to check for the presence of SELECT (either TODD or PRINT) statements in SYSIN at compile time, and to check for a possible loop condition at execution time. End of job will be determined by monitoring the status of the SELECT statements and terminating when there are no more 'active' SELECT statements. 'OFF' indicates that SELECTIT is to bypass all the above checking, and read INPUT1 to end-of-file; this permits a SELECTIT run to count input records without producing any output records. Note: If all SELECTs become inactive during execution, a value of 'ON' will cause SELECTIT to terminate, whereas a value of 'OFF' will force SELECTIT to continue to end-of-file.
PADCHAR(X'40' | 1-byte-literal)	C' ' (Blank)	This parameter defines the character which will be used to fill the remainder of the target buffer when a shorter source buffer is copied to it. It is also used as the pad character when a 'floating' ALTER results in a compression of the record.
PDSMODE(OFF | ON)	OFF	This parameter will activate (ON) or de-activate (OFF) the PDS support in SELECTIT. If activated, SELECTIT will allow processing of entire PDS datasets if requested in the JCL. This OPTIONS keyword MUST be specified via the PARM field, it will not generate an error, but will be ignored if processed on a SYSIN statement. See "Processing PDS Datasets" for details on PDS support.
RETLABEL(RETURN | ¦label)	RETURN	This parameter defines the target label to which control will be passed when the last executable statement in the SYSIN stream is executed but is not a GOTO statement.

		Note: This parameter does not replace the label RETURN; rather, it provides an implicit GOTO at the end of your SYSIN statements.
STATS(ON | OFF)	ON	'ON' indicates that all I/O statistics are to be listed on SYSPRINT. 'OFF' indicates that no statistics are to be listed. If SYSPRINT is absent, STATS(OFF) will be assumed. Note: Non-zero UCTRx values will always be printed, even with LIST(OFF) and STATS(OFF), provided that SYSPRINT is present.
TRACENUM(ON | OFF)	ON	'ON' indicates that SELECTIT is to generate internal data, which will permit the identification of statement and word numbers in the event of an abend. 'OFF' indicates that SELECTIT is to omit the overhead associated with such data - if an abend occurs, only an offset will be available. If any DEBUG statements are present in the SYSIN stream, TRACENUM(ON) will be forced.
UBSIZE(1024 | numeric)	1024	This parameter defines the size (in bytes) of each user buffer. Note that the user buffers are called UBUFFA through UBUFFZ, but only those buffers referenced by SYSIN commands will be created.

Examples of entering OPTIONS

As a PARM: which will be listed on SYSPRINT (with line number 00000) as:

```
PARM='CAPS(OFF) LIST(OFF)'
```

As a SYSIN statement:

```
OPTIONS CAPS(OFF) LIST(OFF).
```

User PARM Data

It is possible to pass data to the SELECTIT program via the PARM field. This could be used, for example, to pass such things as Batch run ID's, selection operands, cut-off dates, etc. to the SELECTIT program.

Coding the data in the JCL PARM field To specify user data use the PARM keyword with the value in parentheses. e.g. to pass the value BATCH4 the EXEC statement would be:

```
//STEP1 EXEC PGM=SELECTIT,PARM='PARM(BATCH4)'
```

or, if other OPTION parameters are already in the PARM

```
//STEP1 EXEC PGM=SELECTIT,PARM='CAPS(OFF) PARM(BATCH4)'
```

If the value contains spaces then the operand itself should be in quotes AND since we're coding on a JCL statement we need to double the quotes. e.g. to pass the 5 character string 'AB CD' to the program the JCL statement would be:

```
//STEP1 EXEC PGM=SELECTIT,PARM='PARM(''AB CD'')'
```

The maximum size of the field is 8 characters.

Retrieving the data in the SELECTIT program To access the PARM data simply use the fieldname of USERPARM. e.g.To move the USERPARM data to positions 11-18 of a record

```
ALTER POS11 COPY USERPARM.
```

Advanced Usage Examples

The following examples show a variety of SELECTIT applications. The techniques shown in these examples should provide you with a good selection of 'building blocks' to enable you to construct complex applications of your own.

Splitting a file into 2 or more files

In this example records from the file LOCATION.HISTORY will be split ¦ into two files based on the PROVINCE field which is in byte 12 of the ¦ record. Provinces with numbers from 1 thru 4 will be written to a file called WESTERN; those with numbers from 5 thru 10 to a file named EASTERN; any with illegal numbers will be dumped to SYSPRINT in character format.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT   DD SYSOUT=*
//INPUT1     DD DSN=LOCATION.HISTORY,DISP=SHR
//WESTERN    DD DSN=LOCATION.WESTERN,DISP=(,KEEP),UNIT=..., etc.
//EASTERN    DD DSN=LOCATION.EASTERN,DISP=(,KEEP),UNIT=..., etc.
//SYSIN      DD *
IF PROVINCE12 > H'0' AND < H'5'
   THEN SELECT TODD(WESTERN) GOTO RETURN.
IF PROVINCE12 > H'4' AND < H'11'
   THEN SELECT TODD(EASTERN) GOTO RETURN.
SELECT PRINT(CHAR).
/*
```

Notes:

- The IF commands take advantage of the implied operand facility. ¦ For example the first IF could have been coded as any of the following:

```
IF PROVINCE12 = H'1' OR H'2' OR H'3' OR H'4' OR H'5' THEN ...
```

or

```
IF PROVINCE12 > H'0' AND PROVINCE12 < H'5' THEN ...
```

or

```
IF PROVINCE12 = H'1' OR
   PROVINCE12 = H'2' OR
   PROVINCE12 = H'3' OR
   PROVINCE12 = H'4' THEN ...
```

- The GOTO RETURN operand on the SELECT statements is needed to prevent the logic from 'falling through' to the next sequential statement. If the GOTO was not coded, all records would end up being dumped on SYSPRINT.

Creating multiple record formats

In this example records from the input file are to be split into two files. The input file is a RECFM=F,LRECL=200 dataset containing a variety of record types (the type is a 1 byte field in position 2). Record types 'H', 'R' and 'X' are to be written unchanged to an output file with the same characteristics as the input file. Record types 'V' and 'W' are to be written to an output file which is RECFM=V.

The 'V' record types are to be written as 35 byte records, the 'W' record types as 22 byte records. All other record types in the input file are to be ignored.

Note: The data in the 'V' and 'W' records is assumed to already be left justified in the input records and does not need any manipulation to prepare it for output.

```
//STEP1   EXEC PGM=SELECTIT,PARM='CBLKSIZE(NO)'
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=INPUT.DATA,DISP=SHR
//FIXEDOUT  DD DSN=FIXED.OUTPUT.FILE,DISP=(,KEEP),UNIT=..., etc.
//VAROUT    DD DSN=VARIABLE OUTPUT.FILE,DISP=(,KEEP),
//             DCB=(RECFM=VB,LRECL=...,BLKSIZE=...),UNIT=..., etc.
//SYSIN     DD *
IF TYPE2 = C'H' OR C'R' OR C'X'
   THEN SELECT TODD(FIXEDOUT) GOTO RETURN.
IF TYPE2 = C'V'
   THEN SELECT TODD(VAROUT) LENGTH(35) GOTO RETURN.
IF TYPE2 = C'W'
   THEN SELECT TODD(VAROUT) LENGTH(22) GOTO RETURN.
/*
```

Notes:

- The VAROUT DD statement must have the DCB parameters specified to prevent SELECTIT from supplying defaults from the INPUT1 file (INPUT1 contains fixed length records, the desired format for the output DD VAROUT is RECFM=V).
- Note that the CBLKSIZE(NO) OPTION has been requested via the PARM field so that if BLKSIZE is not specified in the JCL, the operating system will choose an optimum value for the dataset.
- The SELECT statements which write the records to VAROUT contain the LENGTH operand to specify the desired length of the record. If the LENGTH operand had been omitted, SELECTIT would have written the records as 204 byte records (INPUT1 record length + 4 bytes for the addition of the RDW needed for variable length records).

Combining (merging) multiple input files

In this example two files are to be combined. The input files are both in sequence by customer number (a 5 byte field in location 11). If any duplicate records are detected, the record from the primary input should be copied to the output file and the duplicate record from the secondary file should be dumped to SYSPRINT.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=PRIMARY.INPUT,DISP=SHR
//INPUT2    DD DSN=SECONDARY.INPUT,DISP=SHR
//COMBINED  DD DSN=COMBINED.OUTPUT,DISP=(,KEEP),UNIT=..., etc.
//SYSIN     DD *
OPTIONS AUTOCOPY(OFF).
GETNEXT FILE(INPUT2) EOF(COMPARE).

COMPARE:
IF INPUT1:CUST11 = INPUT2:CUST11/5
   THEN GOTO MATCHED.
IF INPUT1:CUST11 > INPUT2:CUST11/5
   THEN GOTO INPUT2LO.

INPUT1LO:
ALTER OBUFF COPYBUFF(INPUT1).
SELECT TODD(COMBINED).
GETNEXT FILE(INPUT1) EOF(COMPARE).
GOTO COMPARE.

INPUT2LO:
ALTER OBUFF COPYBUFF(INPUT2).
SELECT TODD(COMBINED).
GETNEXT FILE(INPUT2) EOF(COMPARE).
GOTO COMPARE.

MATCHED:
IF INPUT1:CUST11 = 5X'FF'
   THEN GOTO EOJ. ¦
ALTER OBUFF COPYBUFF(INPUT1).
SELECT TODD(COMBINED).
ALTER OBUFF COPYBUFF(INPUT2).
SELECT PRINT(DUMP).
GETNEXT FILE(INPUT1) EOF(MATCHGET).
MATCHGET:
GETNEXT FILE(INPUT2) EOF(COMPARE).
GOTO COMPARE.
```

Notes:

- The OPTIONS AUTOCOPY(OFF) is present to prevent SELECTIT from automatically copying every INPUT1 record to the output buffer (OBUFF) as soon as it is read. The record needed in the output buffer is not known until the INPUT1 and INPUT2 records have been compared.

- The first GETNEXT FILE(INPUT2) ... is required before entering the logic at COMPARE since SELECTIT will only automatically read the first record from INPUT1, before giving control to your commands. Without this GETNEXT there would be no INPUT2 record in the buffer to be compared.

- The test at the beginning of the MATCHED code is to determine when end of job is reached. When end of file is reached on either of the input files, SELECTIT fills the appropriate input buffer with the EOFFILL character (default X'FF'). When both buffers are equal and contain X'FF' in the key, then both files have reached end-of-file.

- All the GETNEXT commands have an EOF(.....) operand coded to basically ignore the condition. Without this operand, end-of-file on either file would have taken the normal default of EOJ and the step would have terminated before processing all the records from the opposite file.

Scanning for 'fixed' strings

An extract from a history log file is desired to review certain activity records. The log file however consists of basically free-form messages. The desired records cannot be located by testing specific locations for specific values. The desired activity records are those to do with transaction ID 'DPX02A' and any to do with userid 'DPMST12'.

```
//LOG      EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=HISTORY.LOG,DISP=SHR
//SYSIN     DD *
IF 0 = C'DPX02A' OR
   0 = C'DPMST12'
   THEN SELECT PRINT(CHAR).
/*
```

Notes:

- The run uses the 'floating' field facilities of SELECTIT.
- The commands IF 0 = ... are basically interpreted as requests to scan the record for the requested strings. If the string is located, the condition evaluates to TRUE, and the record will be printed.

Scanning for 'wild card' strings

This example is very similar to the prior one. The difference is that the desired strings may be numerous but all of a particular format. For example, instead of searching for 'DP001TB', 'DP003TB', 'DP033TB', etc. a search for a string whose 1st 2 characters are 'DP' followed by anything in the next 3 followed by 'TB' may suit your needs. It will certainly be more efficient.

```
//LOG      EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=HISTORY.LOG,DISP=SHR
//SYSIN     DD *
IF 0 MASK C'DP???TB'
   THEN SELECT PRINT(CHAR).
/*
```

Notes:

- The command IF 0 MASK C'DP???TB' requests exactly what is described above. The '?' characters are treated by the IF...MASK as 'I don't care' characters; any characters in these positions are treated as matching characters.

Extracting data from JCL statements

An extract of information about dataset usage is required from a sequential file consisting of a series of batch jobs. The desired output record is to contain 1 record for every DD statement which references a dataset. The record should contain:

```
   Bytes 1-8   JOBNAME
         9-16  STEPNAME
        17-24  DDNAME
        25-68  DSNAME
```

```
//EXTRACT EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=JCL.INPUT,DISP=SHR
//OUTPUT1   DD DSN=EXTRACT.DATA,DISP=(,KEEP),...
//SYSIN     DD *
IF 1 = C'//*' THEN GOTO RETURN.
IF 1 = C'//' AND 0 = C' JOB ' THEN DO.
   ALTER UBUFFA:1 COPY CL68' '.
   ALTER UBUFFA:1 COPY 3/C' '.
   GOTO RETURN.
END.
IF 1 = C'//' AND 0 = C' EXEC ' THEN DO.
   ALTER UBUFFA:9 COPY CL8' '.
   ALTER UBUFFA:9 COPY 3/C' '.
   GOTO RETURN.
END.
IF 1 = C'//' AND 0 = C' DD ' THEN DO.
   IF 3 ¬= C' ' THEN DO.
      ALTER UBUFFA:17 COPY CL8' '.
      ALTER UBUFFA:17 COPY 3/C' '.
   END.
END.
IF 1 = C'//' AND 0 = C'DSN=' THEN DO.
   ALTER UBUFFA:25 COPY CL44' '.
   ALTER UBUFFA:25 COPY 0+4/C' ,('.
   ALTER 1 COPY CL80' '.
   ALTER 1 COPY UBUFFA:1/68.
   SELECT TODD(OUTPUT1).
END.
/*
```

Notes:

- The procedure used here is to utilize the UBUFFA area as a 'staging' area. That is, as the JCL stream is processed, the required information from JOB, EXEC, and DD statements is saved and when a JCL statement which contains DSN= is encountered, the DSN is extracted and the record written.

- The extract for the DSN= value shows the use of a delimited ALTER COPY with multiple delimiters. The statement

  ```
  ALTER UBUFFA:25 COPY 0+4/C' ,('
  ```

  effectively says...

  Copy from 4 bytes past the last successful scan (DSN=) to position 25 of UBUFFA. Stop copying when either a `' '` (blank), `','` (comma), or `'('` (opening parenthesis) is encountered.

Extracting 'words' from a text file

In this example a text file is processed and the desired output is a file containing 1 'word' per record (a 'word' is considered to be any string delimited by blanks). Both files contain variable length records.

```
//EXTRACT EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=TEXT.INPUT,DISP=SHR
//WORDS     DD DSN=WORDS.DATA,DISP=(,KEEP),...
//SYSIN     DD *
OPTIONS AUTOCOPY(OFF).
ALTER UCTRA COPY INPUT1:DATALEN.
ALTER UCTRA SUB F'4'.
ALTER UCTRB COPY F'5'.

NEXTWORD:
ALTER OBUFF:1 COPY X'00040000'.
ALTER UCTRC COPY F'5'.

FINDSTRT:
IF INPUT1:UCTRB = C' ' THEN DO.
   ALTER UCTRB ADD F'1'.
   ALTER UCTRA SUB F'1'.
   IF UCTRA = F'0' THEN GOTO RETURN.
   GOTO FINDSTRT.
END.

COPYLOOP:
IF INPUT1:UCTRB ¬= C' ' THEN DO.
   ALTER OBUFF:UCTRC/1 COPY INPUT1:UCTRB.
   ALTER UCTRC ADD F'1'.
   ALTER UCTRB ADD F'1'.
   ALTER UCTRA SUB F'1'.
   ALTER OBUFF:1/2 ADD H'1'.
   IF UCTRA ¬= F'0' THEN GOTO COPYLOOP
END.

WRITE:
SELECT TODD(WORDS).
IF UCTRA ¬= F'0' THEN GOTO NEXTWORD.
/*
```

Notes:

- Since the output buffer is totally built by the program, an OPTIONS AUTOCOPY(OFF) is present to eliminate the normal copy of the input buffer to the output buffer after it is read.

- Since the input records are variable, the length of the record read is picked up from the INPUT1:DATALEN variable, placed in UCTRA, and adjusted for the RDW so that it reflects 'real' data bytes.
- A pointer to the data bytes is maintained in UCTRB.
- The logic at NEXTWORD initializes the output buffer with a minimum RDW and sets UCTRC to point at the first available output position.
- The FINDSTRT routine finds the start of the next word in case the UCTRB pointer happens to be pointing at a space. If no non-blank is located, a new record is read from the input file.
- The COPYLOOP routine simply copies byte by byte from the input buffer to the output buffer until a blank or end of record is encountered. As it does this, it updates the length in the RDW so that the correct length record will eventually be written.
- The WRITE routine writes the record and either goes back to NEXTWORD if more data remains in the input buffer, or 'falls through' to RETURN if another record needs to be read.

Building commands for another program

In this example an input record contains 3 fields; DSNAME (bytes 1-44), DEVICE (bytes 45-50), and VOLSER (bytes 51-56). The desired output is to be a SCRATCH command to some other utility. The format of the desired command is:

```
SCRATCH dsname,device,volser
```

where the operands are variable length, separated by commas.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=INPUT.FILE,DISP=SHR
//COMMANDS  DD DSN=...,DISP=(,PASS),UNIT=..., etc.
//SYSIN     DD *
ALTER 1 COPY CL80'SCRATCH'.
ALTER 9 COPY DSNAME1/44.
IF OBUFF:0+9 = C' ' THEN DO.
   ALTER OBUFF:0 COPY C',.
   ALTER OBUFF:0+1/= COPY DEVICE45/6.
   IF OBUFF:0+9 = C' ' THEN DO.
      ALTER OBUFF:0 COPY C',.
      ALTER OBUFF:0+1/= COPY VOLSER51/6.
   END.
END.
SELECT TODD(COMMANDS).
/*
```

Notes:

- The output record is built in the following stages:
  - The word SCRATCH is placed in the buffer and the DSNAME field copied to its right.
  - The end of the DSNAME is searched for (IF OBUFF:0+9 = C' '), and a comma and the DEVICE field placed where the blank was found.
  - The process is repeated again by finding the end of the DEVICE field and then adding the comma and the VOLSER field.
- The search for the end of the DSNAME is coded as IF OBUFF:0+9 requests the scan to start in position 10 of the buffer. If coded as IF OBUFF:0 ... the condition would be satisfied by the blank between the command name SCRATCH and the dsname; not the end of the DSNAME.
- Note that the ALTER OBUFF:0 ... operands must all be subordinate to a 'floating' IF to ensure that a valid location has been set for the move.
- The '/=' length override has been added to the two ALTERs which move in the DEVICE and VOLSER

fields. This is required since the 'IF 0 ... ' only searched for a 1 byte constant (C' '). Since the ALTER is now moving a 6 byte field, SELECTIT would normally treat the move as a record expansion. An error condition would be created since this expansion would increase the record's size past the size of the output buffer. The '/=' operand overrides the expansion and treats the copy as an overlay.

Building 'sentences' from words

In this example an input file consists of records with 1 'word' (i.e. a string terminated by a blank) starting in position 1. The desired output is to be file of 80 character lines with the 'words' added 1 by 1 to form 'sentences'.

```
//STEP1   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=WORD.FILE,DISP=SHR
//SENTENCE  DD DSN=SENTENCE.FILE,DISP=(,KEEP),UNIT=..., etc.
//SYSIN     DD *
OPTIONS AUTOCOPY(OFF).
ALTER 1 COPY CL80' '.
ALTER UCTRL COPY F'80'.
ALTER UCTRP COPY F'1'.

LOOP:
ALTER UBUFFA:1 COPY 1/C' '.
ALTER UCTRW COPY FLEN.
IF UCTRW > UCTRL THEN GOSUB RESETLINE.
ALTER OBUFF:UCTRP/UCTRW COPY 1.
ALTER UCTRW ADD F'1'.
ALTER UCTRP ADD UCTRW.
ALTER UCTRL SUB UCTRW.
IF UCTRP > F'80' THEN GOSUB RESETLINE.
GETNEXT FILE(INPUT1) EOF(LASTLINE).
GOTO LOOP.

LASTLINE:
IF UCTRP ¬= F'1' THEN SELECT TODD(SENTENCE).
GOTO EOJ.

RESETLINE:
SELECT TODD(SENTENCE).
ALTER 1 COPY CL80' '.
ALTER UCTRL COPY F'80'.
ALTER UCTRP COPY F'1'.
GOBACK.
/*
```

Notes:

- Since the creation of the output buffer is totally controlled by the commands, an OPTIONS AUTOCOPY(OFF) is used to prevent the automatic copying of the input record to the output buffer when it is read.

- The following UCTRx variables are used:
  - UCTRL Remaining length in the output line
  - UCTRP Position in the line for the 'next' word
  - UCTRW Length of the current word

- The moves to the output buffer are all performed using dynamic buffer addresses and lengths (via UCTRP and UCTRW).

- The remaining logic is fairly straightforward:
  - (A) The length of the current 'word' is determined by copying it to UBUFFA with a delimited copy (ALTER UBUFFA:1 COPY 1/C' '.) and obtaining the length of the word from the FLEN 0 variable.
  - If there isn't room left in the current line for this word, write out the line and reset the UCTRP and UCTRL variables for a new line. This is performed via the GOSUB RESETLINE.
  - If there is room, add the word to the line and adjust the pointers for the next word (allowing for a blank between words).
  - Check again for line overflow and write and reset the line if needed (using RESETLINE).
  - Get the next input and loop back to (A).
  - At end of file, dump the last line only if some words have already been placed in it.

Using UCTRS to accumulate statistics

An input file contains two record types as follows:

```
Type 'A' Byte 1     = C'A'
           "  10-13 = Quantity (binary counter)
Type 'B'   "  1     = C'B'
           "  10-13 = Quantity (binary counter)
```

The desired output is a count of the type 'A' and 'B' records and separate accumulations of the quantity fields for each record type.

```
//COUNT   EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=DATA.FILE,DISP=SHR
//SYSIN     DD *
OPTIONS OPCHECK(OFF).
EQUATE TYPEA# UCTRA.
EQUATE TYPEB# UCTRB.
EQUATE TYPEAQ UCTRC.
EQUATE TYPEBQ UCTRD.

IF RECTYPE1 = C'A' THEN DO.
   ALTER TYPEA# ADD F'1'.
   ALTER TYPEAQ ADD QUANTITY10/4.
   GOTO RETURN.
END.
IF RECTYPE1 = C'B' THEN DO.
   ALTER TYPEB# ADD F'1'.
   ALTER TYPEBQ ADD QUANTITY10/4.
END.
/*
```

Notes:

- The OPTIONS OPCHECK(OFF) command is required since this run has no SELECT commands to produce output. i.e. no records are being written to output files nor are any records being printed. SELECTIT would normally diagnose this as an error condition during the 'compile' phase and would terminate the run.
- For this run, all useful output will be in the statistics written to SYSPRINT at end of job.
- The UCTRx usage is:
  - UCTRA Count of type 'A' records read

- UCTRB Count of type 'B' records read
- UCTRC Accumulated quantity field from all type 'A' records
- UCTRD Accumulated quantity field from all type 'B' records

- This example shows a simple use of EQUATE commands to define the UCTRs used in terms of their actual usage. In coding a more extensive program this could avoid the 'Which counter is used for what?' type of error.

Simple report with page headings

The input records contain 3 character format fields; PARTNO (bytes 1-5), DESCRIPT (bytes 6-20), and LOCATION (bytes 24-30); and a Packed decimal field QUANTITY (bytes 21-23). A simple listing with page headings is desired.

```
//REPORT  EXEC PGM=SELECTIT,PARM='PARM(DRAFT)'
//INPUT1    DD DSN=INVENTORY.FILE,DISP=SHR
//REPORT    DD SYSOUT=*,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
//SYSIN     DD *
OPTIONS AUTOCOPY(OFF).
EQUATE IPARTNO INPUT1:1/5. EQUATE OPARTNO OBUFF:2.
EQUATE IDESCR INPUT1:6/15. EQUATE ODESCR OBUFF:8.
EQUATE IQUANT INPUT1:21/3. EQUATE OQUANT OBUFF:25.
EQUATE ILOCATE INPUT1:24/7. EQUATE OLOCATE OBUFF:31.
EQUATE PAGECTR UCTRP.
EQUATE LINECTR UCTRL.

IF LINECTR = F'0' THEN DO.
   ALTER 1 COPY CL133'1I N V E N T O R Y    L I S T    PAGE:'.
   ALTER PAGECTR ADD F'1'.
   ALTER 39/3 EDITB PAGECTR.
   ALTER 43 COPY USERPARM.
   SELECT TODD(REPORT).
   ALTER 1 COPY CL133'-PART# DESCRIPTION      Q.O.H. LOCATION'.
   SELECT TODD(REPORT).
   ALTER 1 COPY CL133'0'.
   SELECT TODD(REPORT).
   ALTER LINECTR COPY F'55'.
END.

ALTER 1 COPY CL133' '.
ALTER OPARTNO COPY IPARTNO.
ALTER ODESCR COPY IDESCR.
ALTER OQUANT EDITP IQUANT.
ALTER OLOCATE COPY ILOCATE.
SELECT TODD(REPORT).
ALTER LINECTR SUB F'1'.
```

Notes:

- The OPTIONS AUTOCOPY(OFF). is present to stop the automatic copying of each input recrd to the output buffer. Since the output buffer consists only of print lines formated by the remaining statements, the copy is simply wasteful.

- This example shows the use of EQUATE commands to define the layout of the input record fields used, the locations of the fields in the print line, and alternate definitions for the UCTRx variables used as page and line counters.

- The UCTRx variables are used as follows:
  - UCTRL (LINECTR) Line counter to control page overflow
  - UCTRP (PAGECTR) Page counter

- The first conditional block (IF PAGECTR = 0 THEN...) prints the page headings and resets the line count. Since all UCTRx variables are initialized by SELECTIT to zero, this code will be invoked on the first record to ensure that the report starts off with correct headings. Note the use of USERPARM to insert text into the page heading from the external JCL EXEC PARM field.

- The second block of instructions simply moves the data to the appropriate positions of the print line and writes the line to the REPORT file.

Report with control breaks and totals

The input records contains 3 fields; STORENO (bytes 1-5, char.), TRANSNO (bytes 6-8, char), and AMOUNT (bytes 9-12, binary). The file is sorted by STORENO and control breaks and sub-totals are desired whenever the STORENO changes.

```
//REPORT EXEC PGM=SELECTIT
//INPUT1   DD DSN=SORTED.INPUT,DISP=SHR
//OUTPUT1  DD SYSOUT=*,DCB=(RECFM=FA,LRECL=133,BLKSIZE=133)
//SYSIN    DD *
OPTIONS EOFLABEL(TOTALS) UBSIZE(5).
IF CREC = F'1' ALTER UBUFFA:1 COPY STORENO1/5.

IF STORENO1 ¬= UBUFFA:1/5 THEN DO.
   ALTER 1 COPY CL133'0SUBTOTAL'.
   ALTER 22/9 EDITB$ UCTRS.
   SELECT TODD(OUTPUT1).
   ALTER UBUFFA:1 COPY STORENO1/5.
   ALTER UCTRT ADD UCTRS. ALTER UCTRS COPY F'0'.
END.

IF UCTRL = F'0' THEN GOSUB PAGEHEADS.
ALTER 1 COPY CL133' '. ALTER 2 COPY STORENO1/5.
ALTER 13 COPY TRANSNO6/3. ALTER 22/9 EDITB$ AMOUNT9/4.
SELECT TODD(OUTPUT1).
ALTER UCTRL SUB F'1'.
ALTER UCTRS ADD AMOUNT9/4.
GOTO RETURN.

PAGEHEADS:
ALTER 1 COPY CL133'1D E T A I L L I S T I N G'.
SELECT TODD(OUTPUT1).
ALTER 1 COPY CL133'-STORENO TRANSNO AMOUNT'.
SELECT TODD(OUTPUT1).
ALTER 1 COPY CL133'0'.
SELECT TODD(OUTPUT1).
ALTER UCTRL COPY F'55'.
GOBACK.

TOTALS:
ALTER 1 COPY CL133'0SUBTOTAL'.
ALTER 22/7 EDITB$ UCTRS. ALTER UCTRT ADD UCTRS.
SELECT TODD(OUTPUT1).
ALTER 1 COPY CL133'0FINAL TOTAL'.
ALTER 22/7 EDITB$ UCTRT.
SELECT TODD(OUTPUT1).
```

```
GOTO EOJ.
```

Notes:

- The OPTIONS EOFLABEL(TOTALS) is needed so that the TOTALS routine can be invoked to print the final totals.
- UBUFFA is used simply as a working area to hold the STORENO of the last record processed. The OPTIONS UBSIZE(5) simply reduces the allocated size from its 1024 default to 5 bytes since that is all that is used.
- The logic controlled by the IF CREC = F'1' at the beginning simply initializes the previous STORENO area so that the 1st record does not cause a wasted control break.
- The UCTRx variables are used as follows:
  - UCTRL Line counter to control page overflow.
  - UCTRS Counter to accumulate AMOUNT at the store level.
  - UCTRT Counter to accumulate the final total for all stores.

Controlling OPEN/CLOSE of files

In this example 3 input files are to be read and selected records written to 3 output files. The output files are to be created as files 1, 2, and 3 on a single tape volume. The desired records from each file are those with the characters 'DORMANT' in bytes 3-9 of each record.

```
//EXTRACT EXEC PGM=SELECTIT
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=DATA1.FILE,DISP=SHR
//INPUT2    DD DSN=DATA2.FILE,DISP=SHR
//INPUT3    DD DSN=DATA3.FILE,DISP=SHR
//TAPE1     DD DSN=DATA1.DORMANT,DISP=(,KEEP),LABEL=(1,SL),
//             UNIT=TAPE,VOL=(,RETAIN)
//TAPE2     DD DSN=DATA2.DORMANT,DISP=(,KEEP),LABEL=(2,SL),
//             UNIT=TAPE,VOL=(,RETAIN,REF=*.TAPE1)
//TAPE3     DD DSN=DATA3.DORMANT,DISP=(,KEEP),LABEL=(3,SL),
//             UNIT=TAPE,VOL=(,RETAIN,REF=*.TAPE1)
//SYSIN     DD *
OPTIONS DEFER(TAPE2) DEFER(TAPE3).

LOOP1:
IF 3 = C'DORMANT' THEN SELECT TODD(TAPE1).
GETNEXT FILE(INPUT1) EOF(END1).
GOTO LOOP1. ¦
END1: CLOSE OUTPUT(TAPE1).

LOOP2:
GETNEXT FILE(INPUT2) EOF(END2) COPYBUFF(YES).
IF INPUT2:3 = C'DORMANT' THEN SELECT TODD(TAPE2).
GOTO LOOP2.
END2: CLOSE OUTPUT(TAPE2).

LOOP3:
GETNEXT FILE(INPUT3) COPYBUFF(YES).
IF INPUT3:3 = C'DORMANT' THEN SELECT TODD(TAPE3).
GOTO LOOP3.
/*
```

Notes:

- Since the TAPE2 and TAPE3 files cannot be OPENed until the TAPE1 file has been written, an OPTIONS statement requests DEFER for those two DD's.

- Each file is copied by its own code loop (LOOP1:, LOOP2:, and LOOP3:) which reads a record, selects the desired records and writes to the appropriate output file. When end-of-file is reached the output file is

CLOSEd to make the tape available for the next output file. Note the minor difference in structure between LOOP1: and the other two. LOOP1: is entered with the first record from INPUT1 already read; the other two are entered with no record having yet been read from the file.

- The last loop (LOOP3:) does not need to CLOSE the TAPE3 file since it is the last file and normal SELECTIT termination will perform that function.

PDS Input Processing

In this example a source library is dumped to a sequential dataset. An IEBUPDTE control card is inserted prior to the data for each member so that it can later be reloaded into a PDS.

```
//PDSDUMP EXEC PGM=SELECTIT,PARM='PDSMODE(ON)'
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=SOURCE.LIBRARY,DISP=SHR
//UNLOAD    DD DSN=UNLOADED.SOURCE,DISP=(,KEEP),UNIT=SYSDA,SPACE=(...)
//SYSIN     DD *
IF CREC = F'1' THEN DO.
   ALTER 1 COPY CL80'./ ADD NAME='.
   ALTER 13/8 COPY IMEMBER.
   SELECT TODD(UNLOAD).
   ALTER OBUFF COPYBUFF(INPUT1).
END.
SELECT TODD(UNLOAD).
/*
```

Notes:

- The PARM='PDSMODE(ON)' activates PDS support; the fact that the INPUT1 DD does NOT reference a member name will trigger PDS mode processing for the INPUT1 DD. The commands present on SYSIN will be executed for EACH member of the SOURCE.LIBRARY dataset.

- The 1st part of the SYSIN processing is to build a ./ ADD control card and insert it prior to the actual member's data. The ALTER OBUFF COPYBUFF(INPUT1) is needed to restore the data for the first card into the output buffer. Building the ./ ADD card had destroyed this data.

PDS Update Processing

In this example a JCL library is to be updated. All references to UNIT=REEL are to be altered to UNIT=CART. The updated members are to be written to a new alternate JCL library.

```
//JCLUPDT EXEC PGM=SELECTIT,PARM='PDSMODE(ON)'
//SYSPRINT  DD SYSOUT=*
//INPUT1    DD DSN=JCL.LIBRARY,DISP=SHR
//OUTPUT1   DD DSN=NEWJCL.LIBRARY,DISP=(,KEEP),UNIT=SYSDA,SPACE=(...)
//SYSIN     DD *
IF 0 = C'UNIT=REEL' THEN ALTER 0 COPY C'UNIT=CART'.
SELECT TODD(OUTPUT1).
/*
```

Notes:

- The PARM field has activated PDS support; neither the INPUT1 nor OUTPUT1 DD have specified member names so PDS processing will be activated for each file. The commands present on SYSIN will be executed for EACH member of the JCL.LIBRARY dataset. Since OUTPUT1 is activated for PDS processing and no alteration of the member name(s) is done by the input program, each output member will have the same member name as the input member from which its data was read.

- The IF statement simply does the conversion of any JCL which contains the UNIT=REEL string to UNIT=CART.

- Note: This is a very simplistic example. If for example, the unitnames had been REEL and CTAPE where the lengths were different, additional logic would be needed to handle overflow through column 72, etc.