Compute (Bridgend) Ltd

SELCOPY C++ (SLC) Language Reference

Release 3.40

8 Merthyr Mawr Road, Bridgend, Wales UK CF31 3NH

Tel: +44 (1656) 65 2222
Fax: +44 (1656) 65 2227

CBL Web Site - http://www.cbl.com

CBL Ref: z:\cd\sman\l340\SELCOPY_C++_3.40_Language_Reference.pdf

Contents

Contents

Contents

Contents

Documentation Notes

Second Edition, July 2017

Information in this document details general features and functionality of the SELCOPY 3.40 C++ program, available on mainframe platforms as component **SLC** of **SELCOPY Product Suite**, and on Windows, iSeries and supported Unix platforms as **SELCOPY**.

SELCOPY for Windows, iSeries and supported Unix operating systems and SELCOPY Product Suite for z/OS, z/VM (CMS) and z/VSE operating systems are available for download and install from **http://www.cbl.com/selcdl.php** .

The following publications for SELCOPY Product Suite and its component products are available in Adobe Acrobat PDF format at **http://www.cbl.com/documentation.php** :

- SELCOPY Product Suite Customisation Guide
- SELCOPY User Manual
- CBLVCAT User Manual
- SELCOPYi Reference and User Guide
- SELCOPYi Text Editor Manual
- SELCOPYi Data Editor (SDE) Manual
- SELCOPYi Training Manual
- SELCOPYi Quick Reference

At the time of publication, this document is believed to be correct. Where the program product differs from that stated herein, Compute (Bridgend) Ltd reserve the right to revise either the program or its documentation at their discretion. CBL do not warrant that upward compatibility will be maintained for any use made of this program product to perform any operation in a manner not documented within the user manual.

The following generic terms are used throughout this document to identify supported operating systems and architecture:

AIX - IBM Power AIX versions.
AS4 - IBM System i OS/400 versions.
CMS - IBM z/VM all versions.
DEC - HP Alpha Tru64 versions 5.1 and later.
HPX - HP PA-Risc HP-UX versions 10.10 and later.
LNX - i386 Linux versions (RHEL, SLES, Ubuntu).
LNZ - zLinux versions (RHEL, SLES).
SUN - Oracle SPARC Solaris versions 2.4 and later.
VSE - IBM z/VSE all versions.
WIN - Microsoft Windows 32-bit versions (XP, Vista, 7 or 8).
ZOS - IBM z/OS all versions.

Introduction

This chapter provides an introduction to SELCOPY (C++ version) batch executable and its possible applications.

About this Book

This book is a language reference for programs written in the SELCOPY language and processed by the following Compute (Bridgend) products:

- SELCOPY Product Suite, SLC for z/OS
- SELCOPY Product Suite, SLC for z/VM CMS
- SELCOPY for Windows (32-bit)
- SELCOPY for Linux (i386)
- SELCOPY for Linux (System z)
- SELCOPY for OS/400
- SELCOPY for AIX
- SELCOPY for Solaris (SPARC)
- SELCOPY for HP-UX (PA-Risc)
- SELCOPY for Tru64 (Alpha)

It provides all the relevant information and guidance required to develop a SELCOPY program.

SELCOPY Overview

This section provides a general overview and history of the SELCOPY software product.

SELCOPY Background

The name SELCOPY is derived from data SELect and COPY, a general description of some of the earliest applications to be written using the software.

SELCOPY was first developed in the BAL (Basic Assembler Language) on IBM mainframe platforms for OS and DOS operating systems, known today as z/OS, z/VSE and z/VM CMS. The BAL version of SELCOPY was first made generally available in 1971. Since this time, it has been actively developed and is currently operational in many mainframe installations worldwide.

Development for SELCOPY written in C++ was started in 1994 for portability across IBM mainframe, AS/400 and the various PC and Unix platforms. Its specification was based on that of SELCOPY written in BAL and so, with only a few exceptions, supports the same syntax. The first release of SELCOPY written in C++ was made generally available in 1996 for PC-DOS/MS DOS, with versions for Microsoft Windows, AS/400, Linux and other Unix operating systems following thereafter.

Since its first release, the C++ version of SELCOPY has introduced many new facilities that are not available in the BAL version. In 2011, so that programmers on mainframe systems could take advantage of these new facilities, the C++ version of SELCOPY was compiled for z/OS and z/VM CMS systems and included as an executable module, SLC, within the SELCOPY Product Suite package for each system. The executable module, SELCOPY, continues to be included within the SELCOPY Product Suite and remains the default SELCOPY language interpreter for legacy programs.

SELCOPY Development

SELCOPY development and support on various operating platforms has primarily been, and continues to be, driven by user requests. Consequently, new releases and build levels have been made generally available only for current operating systems and only for operating systems on which SELCOPY programs are actively being developed. e.g. SELCOPY for OS/2 is no longer developed.

If you are interested in developing new or existing SELCOPY programs and the SELCOPY executable for the system on which you operate is not currently supported or is at a back level, please contact Compute (Bridgend).

Similarly, any comments, new feature requests or software defect reports may be emailed to Compute (Bridgend) at: support@cbl.com

The SELCOPY Language

SELCOPY is a high level, interpreted language and so a SELCOPY program requires no compilation before execution.

Like Assembler, C and COBOL, SELCOPY is a non-structured, procedural programming language which consists of sequentially ordered statements and supports labels allowing execution flow to jump to any location within the program.

The language itself is designed to be practical and easy to understand so that inexperienced programmers may deduce the function of a SELCOPY application without in-depth analysis.

The SELCOPY (or SLC) executable accepts the SELCOPY program as control input, interprets all control statements in order to translate them into SELCOPY internal structures, then executes the statements in order of sequence.

Run-time Environment

Run-time environment pre-requisites exist for C++ SELCOPY executed in the following operating systems.

IBM z/OS

On mainframe z/OS systems, the SELCOPY C++ (SLC) object modules are compiled using the XL C/C++ compiler which requires Language Environment. Language Environment is an IBM supplied architecture that is integral to modern z/OS operating systems and establishes the run-time environment required to run the SLC executable.

SLC can share the run-time environment with other called executables written in languages that conform to the Language Environment architecture.

Microsoft Windows

Required only for SELCOPY support of Micro Focus VSAM files, a folder containing the following run-time (dynamic link) libraries must exist in the PATH environment variable definition.

◊ CBLRTSS.DLL
◊ MFFH.DLL

Input/Output of VSAM (Virtual Storage Access Method) files, a data format native to IBM mainframes, is supported by SELCOPY on Windows platforms via the suite of products provided by Micro Focus International plc.

SELCOPY Applications

A SELCOPY program may be written to perform any number of tasks but is primarily associated with the manipulation of data read from, and subsequently written to, any number and combination of files, data streams or database tables.

Some common uses include:

- Data modification and verification steps within production z/OS batch jobs.
- Interrogation and reformat of data as part of a CLIST or Rexx procedure, Unix script, Windows VBScript or batch file execution.
- Quick, execute once only jobs to trouble shoot and fix data.

Support to seamlessly perform file I/O on multiple, potentially different file formats using simple syntax makes writing a SELCOPY program a flexible, quick and easy alternative to using system utilities (e.g. sed, grep, awk or z/OS IEBCOPY, IEBGENER).

Maintenance

Corrections to software defects and introduction of new functionality are included in C++ SELCOPY as new builds of the current, generally available release.

Update of SELCOPY to a new build level involves replacement of the SELCOPY executable which, for z/OS systems, is achieved via an SMP/E SYSMOD to the SELCOPY Product Suite and, for z/VM CMS systems, via a VMARC archive file extraction. For all other operating systems, the install process must be repeated for the updated SELCOPY product package.

Build levels are incremental. The current build level number of a C++ SELCOPY release follows a period (.) separator which is a suffix to the release number displayed in the SELCOPY list output footer text. e.g. The following footer records are written by SELCOPY for Windows Release 3.20 Build level 001.

```
** SELCOPY/WNT 3.20.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 17 Jul 2014 **
```

The build level is also included in the SELCOPY version output, displayed when input parameter -V is passed to the SELCOPY executable.

Notation Conventions

The following list defines notations used in this publication.

- Text in syntax diagrams and examples of SELCOPY syntax are presented in a `monospace` font.
- SELCOPY keyword identifiers appear in upper case (e.g. READ, WRITE, IF) although may be entered in upper or lower case, or a mixture of cases.
- SELCOPY keyword identifiers representing operation parameters may have trailing, lower case characters. The upper cased portion of the keyword identifies the minimum abbreviation for the keyword. (e.g. Vchar indicates that V, VC, VCH, VCHA and VCHAR are all acceptable alternatives)
- Variables appear in lowercase *italics* (e.g. *field_nATp*) and represent programmer defined parameters or keyword parameter values.
- Syntax diagram footnote references are represented by a number in parentheses (e.g. (1)).
- A single blank may be represented by character b.

Syntax diagrams adhere to the following standards:

Arrow Symbols

Diagrams should be read from left to right, top to bottom and follow the path of the line. Junctions in the line are represented by a plus (+) symbol.

- ◊ `>>-` indicates the beginning of a statement.
- ◊ `->` indicates the statement syntax continues on the next line.
- ◊ `>-` indicates the statement syntax has been continued from the previous line.
- ◊ `-><` indicates the end of a statement.

The horizontal path line, delimited by arrow symbols, denotes the main path of the syntax diagram.

Required Items

Required items appear on the main path.

```
>>-- REQUIRED_ITEM ---------------------------------><
```

Optional Items

Optional items appear below the main path.

```
>>-- REQUIRED_ITEM ---+-----------------------+-----><
                      |                       |
                      +-- optional_item ------+
```

If an optional item appears above the main path, then that item has no effect on the execution of the statement and is used only for readability.

```
                      +-- optional_item ------+
                      |                       |
>>-- REQUIRED_ITEM ---+-----------------------+-----><
```

Multiple Required or Optional Items

If one or more alternative optional items exist, they appear vertically on separate paths. If selection of one of the items is optional, the items appear in paths below the main path.

```
>>-- REQUIRED_ITEM ---+-----------------------+-----><
                      |                       |
                      +-- optional_choice1 --+
                      |                       |
                      +-- optional_choice2 --+
```

If selection of one of the items is mandatory, one of the items appears on the main path and all other items appear on paths below the main path.

```
>>-- REQUIRED_ITEM ---+-- required_choice1 --+-----><
                      |                       |
                      +-- required_choice2 --+
```

Repeatable Items

An arrow occurring above a path line returning to a junction to the left of another junction indicates the item between the junctions may be repeated.

```
                      +----------------------+
                      v                      |
>>-- REQUIRED_ITEM ---+-- repeatable_item ---+-----><
```

If the arrow occurs above a number of multiple item paths, this indicates that more than one of the items may be specified and each of the items are repeatable.

Default Items

If one of a number of optional items is default, the path containing that item appears above the main path.

```
                      +-- default_choice ----+
                      |                      |
>>-- REQUIRED_ITEM ---+----------------------+-----><
                      |                      |
                      +-- optional_choice ---+
                      |                      |
                      +-- optional_choice ---+
```

Fragments

Syntax diagrams may be split into fragments so that related syntax items are removed from the diagram and displayed in a syntax diagram fragment below.

Fragments are named and their location within the parent diagram represented by the fragment name in **bold** print, enclosed in vertical bars (or symbols). The same vertical bars are used to indicate the beginning and end of the syntax diagram fragment below the parent diagram.

```
>>-- REQUIRED_ITEM --- item1 --------| fragment |-->< 

fragment:

|---+-- item2 --+--- KEYWORD --+-----------+-------|
    |           |               |           |
    +-- item3 --+               +-- item4 --+
                                |           |
                                +-- item5 --+
```

Summary of Changes

This section is a summary of new features included in different releases of C++ SELCOPY.

Release 3.20 Enhancements

Release 3.20 includes the following product enhancements:

- In the print control block of SELCOPY's output report, display the offset from the start of the field of printed data that streams onto second and subsequent print lines.
- **z/OS only:** Use RECFM of the input file (data set) as the default for output HFS or ZFS files as opposed to RECFM=U EOL=LF.
- **z/OS only:** As for the BAL version of SELCOPY, the length value displayed against an output file in the summary block of SELCOPY's output report is that of longest record written. Previously, the defined LRECL value was displayed.
- Support new operation CHANGE.
- Support multiple field specifications on WRITE and MOD operations.
- **Windows only:** Support WRITE of keyboard keystrokes to other opened windows.
- **Windows only:** Support READ and WRITE of data to and from the Windows clipboard.

Release 3.30 Enhancements

Release 3.30 includes the following product enhancements:

- Regular expressions supported as search strings on IF/AND/OR and CHANGE operations.
- Support for variable length character data types on the DECLARE operation.
- Support MATCHLEN parameter on IF/AND/OR operations to assign the length of matched text to an @variable or declared variable following a character or regular expression compare.
- Support CASEI parameter on the CHANGE operation to ignore alpha character case in the search string specification.
- **Windows only:** Support WIN RESET on the WRITE operation for Windows key stroke output.

Release 3.40 Enhancements

Release 3.40 includes the following product enhancements:

- **z/OS and z/VM CMS only:** Support debug via the SELCOPY Debug utility of SELCOPYi (a component of the SELCOPY Product Suite). See the ***"SELCOPYi Reference and User Guide"*** for details.
- SELCOPY build date included in the list output footer information.
- Additional SORTDIR codes supported to allow alternative (ascending/descending) input directory sort sequences based on the date, size, name, extension or path fields returned by a READ DIR/DIRDATA operation.
- Declared variables of data type CHARZ are now initialised as null length strings if the INI has not been specified on the DECLARE operation.

Chapter 1. Program Elements

This chapter describes the basic elements used to write a SELCOPY program.

Character Set

SELCOPY supports only syntax written in single byte character set (SBCS) that conforms to the invariant (syntactic) character set 0640. A character belonging to character set 0640 has the same code point in all code pages. This is true of all ASCII (ISO) code pages and most EBCDIC code pages.

SELCOPY C++ source is written using code page 819 (ISO-8859-1 Western Europe) and converted to code page 285 (EBCDIC UK) for mainframe compilation. Therefore, by default, SELCOPY assumes that characters are at code points defined by these code pages.

Code pages 819 and 285 contain the English alphabet characters, 10 decimal digits, special characters and other national language and control characters. SELCOPY constants, programmer-defined names and comment text may contain characters at any SBCS code point. Other SELCOPY syntax elements (e.g. keywords, delimiters and operators) are limited to alphanumeric characters and certain special characters only. These are detailed in the following sections.

Alphabetic Characters

The 26 upper case and 26 lower case alphabetic characters that comprise the English alphabet.

Character	EBCDIC Hex	ASCII Hex		Character	EBCDIC Hex	ASCII Hex
A	C1	41		a	81	61
B	C2	42		b	82	62
C	C3	43		c	83	63
D	C4	44		d	84	64
E	C5	45		e	85	65
F	C6	46		f	86	66
G	C7	47		g	87	67
H	C8	48		h	88	68
I	C9	49		i	89	69
J	D1	4A		j	91	6A
K	D2	4B		k	92	6B
L	D3	4C		l	93	6C
M	D4	4D		m	94	6D
N	D5	4E		n	95	6E
O	D6	4F		o	96	6F
P	D7	50		p	97	70
Q	D8	51		q	98	71
R	D9	52		r	99	72
S	E2	53		s	A2	73
T	E3	54		t	A3	74
U	E4	55		u	A4	75
V	E5	56		v	A5	76
W	E6	57		w	A6	77
X	E7	58		x	A7	78
Y	E8	59		y	A8	79
Z	E9	5A		z	A9	7A

Numerical Characters

The 10 numerical characters.

Character	EBCDIC Hex	ASCII Hex		Character	EBCDIC Hex	ASCII Hex
0	F0	30		5	F5	35
1	F1	31		6	F6	36
2	F2	32		7	F7	37
3	F3	33		8	F8	38
4	F4	34		9	F9	39

Decimal Digits

SELCOPY recognises decimal digits written using numerical characters, 0 through 9. They are referred to simply as digits and are used to write numerical arguments and constants in decimal notation.

Hexadecimal Digits

SELCOPY recognises 16 hexadecimal digits written using numerical characters, 0 through 9 and alphabetic characters, A through F, which altogether represent decimal values 0 through 15. They are referred to as hex digits and are used to write numerical constants or character string constants in hexadecimal notation.

Hex digits may be specified using upper and/or lower case characters. i.e. digits A,B,C,D,E,F may be written as: a,b,c,d,e,f.

Special Characters

The following table identifies non-alphanumeric characters that can have special significance in SELCOPY syntax.

Character	Description	EBCDIC Hex	ASCII Hex
b	Blank	40	20
! (1)	Exclamation mark	5A	21
" (1)	Quotation mark	7F	22
%	Percent	6C	25
&	Ampersand	50	26
'	Apostrophe	7D	27
(	Left parenthesis	4D	28
)	Right parenthesis	5D	29
*	Asterisk	5C	2A
+	Plus sign	4E	2B
,	Comma	6B	2C
-	Minus sign	60	2D
.	Period / Decimal Point	4B	2E
/	Slash	61	2F
:	Colon	7A	3A
<	Less than symbol	4C	3C
=	Equals sign	7E	3D
>	Greater than symbol	6E	3E
@ (1)	Commercial at	7C	40
\ (1)	Backslash	E0	5C
^ (1)	Circumflex accent	BA	5E
¬ (1)	Not sign	5F	AC

(1) The exclamation mark (!), quotation mark ("), commercial at (@), backslash (\), Circumflex (^) and Not sign (¬) have variant EBCDIC code points whereas, in ASCII code pages, only the Not sign (¬) is a variant code point.

Composite Symbols

Special characters may be combined, with no intervening blank delimiters, to create composite symbols. The following table identifies valid composite symbols and their meaning.

Character	Meaning
<> or ^< or ¬=	Not equal to
<= or =<	Less than or equal to
>= or =>	Greater than or equal to
^> or ¬>	Not greater than
^< or ¬<	Not less than
*<	Comment ignoring statement separator
*>	Comment to appear in output summary
/*	End of program input

Case Sensitivity

Any combination of uppercase and lowercase alphabetic characters may be used in SELCOPY control statement syntax.

Lowercase characters used in most identifiers are treated as being their uppercase equivalents. Exception are characters in quoted character constants, comment text and Windows, Unix, OpenVM or OpenMVS style fileid identifiers, where lowercase characters are treated as being lowercase.

Statement Elements

A SELCOPY program is comprised of a number of control statements.

SELCOPY statement elements are categorised as delimiters, identifiers, constants, operators or comment text.

Delimiters

The following table shows delimiter characters that may be used to separate identifiers and constants.

Delimiter	Name	Usage
b , = (1)	Blank Comma Equals sign	A blank delimiter used to separate statement elements.
!	Exclamation mark	Statement separation character.
( )	Parentheses	Used to enclose length or precision/scale specification for declared variables.
:	Colon	If the last character of the first statement identifier, denotes the identifier as being a label name.
\	Backslash	If the last non-blank character of a control file record, indicates statement continues on the next control file record.

(1) The equals sign is a blank delimiter unless it is part of a composite symbol.

Delimiter characters may be used in other contexts. For example Backslash is the control file statement continuation character if the last non-blank character of a record, but may also be used to connect the file path directory elements in a Windows or Unix style file identifier. (e.g. `\\MyPC\c\tmp\Myfile.txt` and `\usr\home\usera\sample.txt`).

With the exception of colon (:), all delimiter characters may be surrounded by one or more of the blank delimiter characters ("b", "," or "="). Any number of blank delimiters may be specified where one blank delimiter is supported. e.g.

```
DECLARE  INREC  CHAR (256  )
```

A colon, when used as label name delimiter, must immediately follow the label name and must be followed by at least one blank delimiter.

Except where specified in a character string constant, multiple consecutive blank delimiter characters are equivalent to a single blank delimiter character.

Identifiers

Identifiers are a series of characters that are not part of a comment or a constant.

Identifiers can be SELCOPY keywords or programmer-defined names.

The first character of an identifier must be preceded by a delimiter or occupy the first character of a control file record. Similarly the last character of an identifier must be followed by a delimiter or occupy the last character of a control file record to be processed by SELCOPY. Exceptions to this rule are environment variables and parameter variables which may be preceded or followed by any character.

A programmer defined name identifier that represents a numeric value may also be preceded or followed by an arithmetic operator when specified as a term in an expression.

One or more of the blank delimiter characters must separate identifiers that are not separated by some other delimiter, arithmetic operator or control file record boundary.

Programmer-defined names that represent a variable or label cannot be the same as a keyword identifier that represents one of SELCOPY's operation keywords that can be specified without a parameter keyword. In all other circumstances, SELCOPY can determine whether or not the identifier is a keyword, in which case programmer-defined names may be the same as a keyword identifier.

SELCOPY Keywords

Keyword identifiers are character strings that have special meaning to the SELCOPY program. Keywords may represent operations that instruct SELCOPY to perform some action, or parameters to an operation.

Programmer Defined Names

Programmer-defined name identifiers are character strings that are used as variables, substitution variables, labels, filenames and fileids.

A programmer-defined name may contain any character, though support for special characters depends upon the type of programmer-defined name and the character's significance to SELCOPY. i.e. a programmer-defined name must not include a delimiter or operator character where it may be interpreted as such. e.g.

- Any blank delimiter will denote a new identifier.
- Statement continuation will occur if the name has a trailing backslash (\) character and is the last statement identifier.
- An arithmetic expression may be compromised if it involves a programmer-defined name that contains plus (+) or minus (-) symbols.

Because of this, it is generally advised that special characters should not be used in programmer-defined names. Furthermore, programmer-defined names that conflict with SELCOPY keyword identifiers (or keyword identifier synonyms) should be avoided.

The different types of programmer-defined names and their notation rules are as follow.

Internal Variables

Variables defined by SELCOPY for general use in SELCOPY program statements. The names of these variables are pre-determined. e.g. RETCD, LINE and LRECL

Declared Variables

Variables defined by the programmer using the DECLARE operation or generated by SELCOPY for each column of data returned by ODBC or SELCOPYi list input.

Apart from restrictions applicable to all programmer-defined names, declared variable names may be of any length and can contain any character with the following exceptions:

1. The name must not contain apostrophes (') or quotation marks (").
2. The first character must not be special character ampersand (&).
3. The first character must not be a decimal digit (0 to 9).
4. The first character must not be special character percent (%) if all remaining characters are decimal digits.

A declared variable must not have the same name as an internal variable or an operation keyword. If so, no error occurs but the identifier will be interpreted as referencing the internal variable or, if the first identifier in the statement, the operation keyword.

@Variables

Integer value variables that have names of any length but must begin with the special character, commercial at (@).

Environment Variables

Variables that have been set by the operating system environment. i.e. Windows and Unix Environment Variables, z/OS and CMS REXX variables and, in z/OS TSO only, z/OS system symbols.

An environment variable name comprises the name of the system environment variable enclosed within percent symbols (%) with no intervening delimiter characters. e.g. %USERNAME% and %SYSNAME%

Alternatively, for z/OS systems only, the environment variable may be expressed as the variable name prefixed by an ampersand (&) and suffixed by a period (.) with no intervening delimiters. e.g. &SYSNAME.

Unlike other identifiers, environment variable names do not need to be enclosed by delimiter characters or the limits of the control file record. e.g. 'user%USERNAME%#001'

Parameter Variables

Variables that represent parameter numbers 0 to 9 passed to the SELCOPY program, where parameter number 0 is the SELCOPY executable program name.

A parameter variable name is comprised of a single decimal digit that corresponds to the parameter sequence number, prefixed by the percent symbol with no intervening delimiter characters. e.g. %3 corresponds to the 3rd input parameter.

Unlike other identifiers, parameter variable names do not need to be enclosed by delimiter characters or the limits of the control file record. e.g. 'Parm%3:'

Equated Symbols

Equated symbols are names defined by the programmer using the EQU operation and represent an equated value. Each occurrence of an equated symbol in control statements that follow the EQU operation will be substituted with the equated value.

An equated symbol name can be of any length.

Labels

A label must be the first identifier in a statement and represents a location within the SELCOPY statements to which processing may be directed via a GOTO or DO operation.

A label name can be of any length but the last character must not be delimiter character colon (:) which may optionally be used to denote the end of a label.

File Names

A file name is 1 to 8 characters in length and is used by SELCOPY to identify an input or output data object.

Fileids

A fileid is the name by which a file or data set is known to the executing operating system. Fileid names are specified on SELCOPY I/O statements for the purpose of associating or dynamically allocating the fileid to a file name.

A fileid name may consist of any character and be of any length and format supported by the executing operating system. Fileids often include special characters that qualify or delimit portions of the fileid name. e.g. colon (:), slash (/), backslash (\), period (.) and parentheses (( )).

If a fileid name contains one of SELCOPY's blank delimiter characters, quotation marks, apostrophes or the separator character, then it should be enclosed in quotations marks (") or apostrophes (') as appropriate.

Constants

Constants are a series of characters that represent a value with an implied data type and length.

One or more of the blank delimiter characters must separate constants that are not separated by some other delimiter, arithmetic operator or control file record boundary.

The value, data type and length of a constant are established by SELCOPY during control statement analysis processing and cannot subsequently be changed by the programmer. See Data Elements and References for details on specification of constant values.

Operators

The following table shows operators that may be used to separate identifiers and constants. Operators may be single characters, composite symbols or SELCOPY keywords.

Operator Type	Operator	Description
Arithmetic (Constants/Expressions)	+	Add or unary plus
	-	Subtract or unary minus
Relational (Conditions)	= b , EQ EX EXACT	Equal to
	<> ^ ^= ¬ ¬= NE NOT	Not Equal to
	< LT	Less than
	> GT	Greater than
	<= =< ^> ¬> LE NGT HI HIGH	Less than or equal to. Not greater than. High limit.
	>= => ^< ¬< GE NLT LO LOW	Greater than or equal to. Not less than. Low limit.
Bitwise (Conditions)	ONES	Bits selected by the mask are all on.
	ZEROS ZEROES	Bits selected by the mask are all off.
	MIXED	Bits selected by the mask are not all on or all off.
Logical (Data Modification)	OR	Inclusive or. Bit is on if either of the bits are on. Otherwise bit is off.
	XOR	Exclusive or. Bit is on if either (but not both) of the bits are on. Otherwise bit is off.
	AND	And. Bit is on if both of the bits are on. Otherwise bit is off.

With the exception of the arithmetic operators plus (+) and minus (-) and relational operators equals (=), blank (b) and comma (,), all operator must be surrounded by one or more of the blank delimiter character. e.g. The following statements are valid:

```
IF  POS 23 LEN 4 TYPE=B >= 245
  THEN @ARR=DVAR-245
  THEN @VAL = DVAR + 20 - @ARR
```

However, the following is invalid:

```
IF @ARR<DVAR
```

Comments

Comments may exist as the whole or part of a SELCOPY statement as a method of providing explanatory notes to programmers reading the statements. The comment data itself is ignored by SELCOPY and does not affect the logic of the control statements.

Comment text begins at the first occurrence within a control file record of the special character asterisk (*) that is not part of a quoted character constant. The asterisk must either be the first character of the statement or be preceded by one of the blank delimiter characters.

Comment text is terminated by an exclamation mark (!) statement separator character (SEP option) that is not part of a quoted character string constant, or the end of a control file record that does not end with the statement continuation character backslash (\). (See statement continuation and separation for details.)

Because of their special significance, comment text may not include the separator character (see comment text ignoring statement separator if this is required) nor end with the statement continuation character in the last non-blank character of the control file record. Otherwise, comment text may contain any character. e.g.

```
**Start of main processing loop.         !DECLARE INREC1 CHAR(80)
READ DDIN1  INTO INREC1     * Input records into variable INREC1.
IF EOF DDIN1   * If no more records.     !THEN EOJ    * End the program.
```

The following two additional forms of comment specification exist that have special significance to SELCOPY.

Comment Text in the Summary Block

Using composite symbol, asterisk greater than (*>), as the start of the comment will display the comment text next to a statement's selection identifier number within the summary block of SELCOPY's output diagnostic report. This may be particularly useful when diagnosing execution of SELCOPY programs that include a large number of control statements.

This type of comment is applicable only on statements that execute an operation other than one of the following:

- A program environment operation.
- An Input/Output operation other than PRINT, LOG and PLOG.
- An IF, OR or AND operation.

If specified on statements that do not match this criterion, the comment will be treated as a regular comment and so will not be displayed in the summary block.

```
IF INCOUNT DDIN1 = 1
  THEN PERFORM  FIRST_RECORD    *> Execute First_Record Subroutine.
```

Comment Text Ignoring Statement Separator

A composite symbol, asterisk less than (*<), as the first delimiter of a statement will treat all text that follows as comment text, ignoring any subsequent statement separator characters. This type of comment is only terminated by the end of a control file record that does not end with the statement continuation character backslash (\).

```
  PRINT V1  * Comment.  !  *< PRINT V2  !PRINT V3   * Only V1 is printed.

*<     Using this type of comment allows \
       the comment data to stream over a \
       number of control file records.
```

Statements

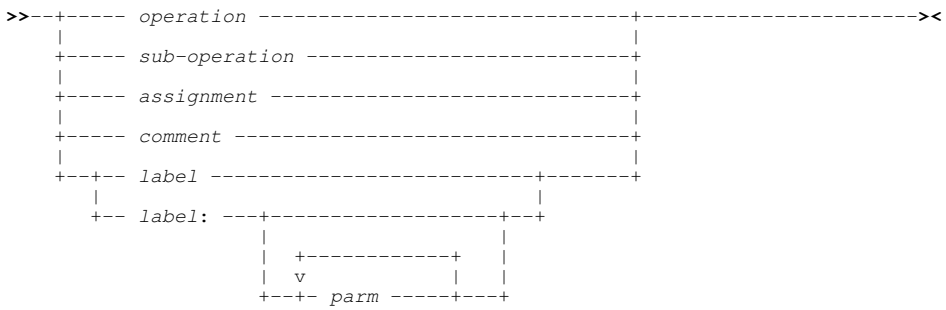
Identifiers, delimiters, operators and constants are used to construct SELCOPY statements.

Null statements and statements containing only blank delimiters are ignored by SELCOPY. They may be entered as aesthetic additions to the control statement file and serve only to increase the spacing between statements displayed in the execution output diagnostic report.

By default, statements are terminated by the end of the control file record in which it occurs. Continuation of statements onto second and subsequent file records is possible by entering the continuation character, backslash (\), as the last non-blank character of the file record.

Similarly, statements may be terminated before the end of the control file record using the statement separation character, exclamation mark (!) In this way, multiple statements may exist on a single control file record.

Non-blank statements have the following format:



operation
: Execute a SELCOPY *operation*. The statement must start with one of SELCOPY's operation keywords.

sub-operation
: A *sub-operation* statement must start with one of SELCOPY's sub-operation keywords. Sub-operations may be one of the following:

◊ A logical sub-operation (AND or OR) which must immediately follow an IF operation or another logical sub-operation.

◊ The conditional sub-operation, THEN, which must immediately follow an IF operation, a logical sub-operation or another conditional sub-operation (THEN or ELSE).

◊ The conditional sub-operation, ELSE, which must immediately follow a THEN sub-operation.

◊ The conditional sub-operation, CAT, which must immediately follow a READ operation.

assignment
An *assignment* statement assigns a value to a variable or a field.

comment
A *comment* statement contains only comment text, denoted by asterisk (*) as the first non-blank character of the statement. See statement elements, comments for details.

label
A *label* identifies a location in the program to which a logic flow operation may direct SELCOPY's processing. A terminating colon (:) is mandatory if the label identifies the start of a sub-routine that is to declare variables (*DCLvar*) and assign them values passed to the sub-routine as parameters.

parm
A sub-routine parameter name to which a value specified on a DO *label* operation will be assigned.

Control Statement File

The SELCOPY executable can accept control statements passed to it as a parameter string or via an input file (data set).

Control statements passed as program parameters are done so as a continuous stream of text with the statement separator character being used to delimit the individual statements. This is documented in more detail under Program Execution.

SELCOPY statements provided in a control file exist on one or more discrete lines (records) belonging to the file. Unless the continuation character is used, the end of a control file record is the natural control statement delimiter.

Hierarchical file systems are native to Windows, Unix and OS/400 operating systems and also exist on z/VM and z/OS systems as CMS BFS and z/OS HFS or ZFS file systems respectively. Control files belonging to a hierarchical file system contain variable length lines that are terminated by end-of-line characters (e.g. LF or CRLF). Control statements may occupy all available characters in this type of control file record.

Control file records belonging to native CMS files and z/OS data sets may be of fixed or variable length (RECFM=F or V).

For RECFM=V format control files, the defined LRECL includes the 4-byte RDW reducing the maximum length of a record by 4. SELCOPY control statements may occupy all available characters of a RECFM=V variable length record up to this maximum length.

For RECFM=F control files, the last 8 characters are reserved for sequence numbers and so are ignored by SELCOPY. Therefore, control statements may occupy all but the last 8 characters of a RECFM=F fixed length record.

End of Program Statements

SELCOPY program control statement input is naturally terminated by the end of the control statement file. Exceptions to this are as follow:

- The end of program indicator is encountered before the last record of the control statement file is read. The end of program indicator is composite symbol (/*) occupying the first two columns of a statement.
- The end of program indicator is passed as a control statement via the SELCOPY program parameters. Control statements passed as program parameters are executed before those passed in the specified control statement file.
- No control statement file is specified, either via the -ctl program parameter or **stdin** (SYSIN for z/OS and z/VM CMS), and no end of program indicator is passed to SELCOPY via the SELCOPY program parameters. In this case, SELCOPY accepts its control statement input via the user's terminal.
- The input control file is a concatenation of a number of files (or data sets). In this case, unless the end of program indicator is encountered first, end of program statements occurs at the last record of the last concatenated file.
- The end of program indicator is encountered within a control statement file that is itself included within supplied control statements via the INCLUDE operation.

Statement Length

SELCOPY control statements have a default maximum length of 4096 characters. This maximum may be extended (but not reduced) using the CONTMAX program environment option.

This limit actually corresponds to the maximum length of a control file record or, if statement continuation is used, concatenation of control file records. It also includes the length of any end-of-line characters. e.g. The default maximum length of a statement starting in column 1 of a CRLF terminated control file record is 4094.

Therefore, the maximum length of a SELCOPY control statement is further reduced when the statement separator character is used to specify multiple statements on the same control file record.

Where statements are provided as parameters to the SELCOPY executable, this maximum applies to the sum of the statement lengths. However, the limit may further be reduced by environment constraints. e.g. Windows XP supports a maximum of 8191 characters entered at the command prompt or via a batch file.

Statement Continuation

SELCOPY statements in control files may span several, consecutive control file records using the statement continuation character, backslash (\).

The statement continuation character must occupy the last non-blank character of each file record over which the statement will be streamed.

```
*...,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....
  READ  EMPTAB  SQL="                                                        \
                     select substr(full_name,1,25) NAME, assignment_id,    \
                                 P.effective_start_date, employee_number   \
                                                                           \
                      from       per_all_people_f        P,              \
                                 per_all_assignments_f   A               \
                                                                           \
                     where       employee_number  >  '7482'              \
                           and   P.person_id      =  A.person_id          \
                    "                * Input Oracle database result table rows.
```

Text belonging to the record following a record ending with the continuation character is joined to the previous record text so that the first character of the second record overlays the continuation character in the first. Therefore, any statement element may be split over the 2 lines. e.g.

```
*...,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....
    IF  POS 301   =  'Character constant text that streams acr\
oss two control file records.'
```

The number of records over which a statement may be streamed is limited only by the maximum length of a statement.

Statement Separation

By default, SELCOPY statements are delimited by the end of a control file record. However, the statement separation character (!) may be used to delimit (separate) SELCOPY statements specified on the same control file record or provided via SELCOPY program parameters.

The default statement separator character is exclamation mark (!). For z/OS and z/VM CMS, this default may be set by the CBLNAME option, Separator. However, on all systems, the default may be changed or disabled altogether, using the SEP program environment option in the SELCNAM file or in the SELCOPY control statements.

The statement separator character is not interpreted as being the statement separation character when specified as part of a quoted character constant or card data input.

Chapter 2. Program Execution

This chapter describes how to execute a SELCOPY program, the processing performed by the SELCOPY executable and the generated output report.

Invoking the Executable

The SELCOPY executable is required to interpret control statements of a SELCOPY program.

On supported mainframe systems (z/OS and z/VM CMS) the name of this executable module is SLC. For all other operating systems, the name of the executable is SELCOPY. For the purpose of this manual, the executable will be referenced as SELCOPY.

This section describes the methods by which SELCOPY may be invoked in each environment.

Exploiting SELCOPY

SELCOPY programs can be general purpose applications that may be exploited from anywhere within the system. Networked file systems are also transparent to SELCOPY programs and so may be used to process data on other systems.

SELCOPY may be executed using methods supported for any executable program. e.g. A SELCOPY program may be started from any of the following:

- A command shell prompt. (e.g. z/VM CMS, z/OS TSO, Unix/Telnet terminal or Windows Command Prompt)
- The Windows "Run" dialog.
- A batch script file.
- A z/OS batch job step.
- A CLIST or REXX procedure.
- Any program written in a language supporting program CALL.

Locating SELCOPY

All operating systems have a method for dynamically locating a program executable if no specific location is provided as part of its execution.

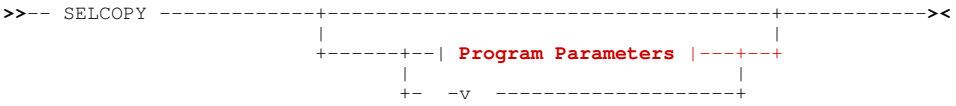
In a Windows or Unix environment the current working directory is searched before searching directories specified by the PATH environment variable. In OS/400 the library list (*LIBL) is searched, in z/OS it is the standard program search path including the active Link List, and in z/VM CMS all accessed mini-disks are searched.

For efficiency, the SELCOPY executable should be installed in a directory or library that is included in the default program search path. The remainder of this publication assumes that this is the case.

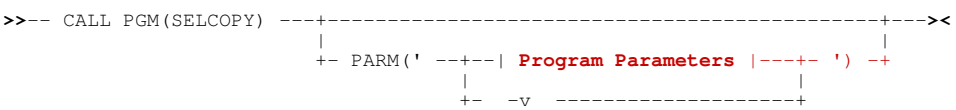
SELCOPY Command

All SELCOPY command program parameter keywords, including -v, may be entered in any character case. e.g. -v is the same as -V.

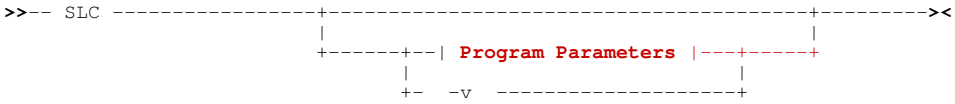
Microsoft Windows and Linux/Unix Systems:



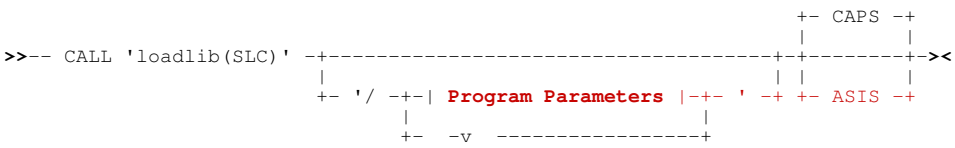
OS/400:



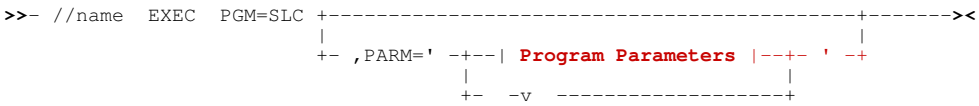
z/OS TSO and z/VM CMS:



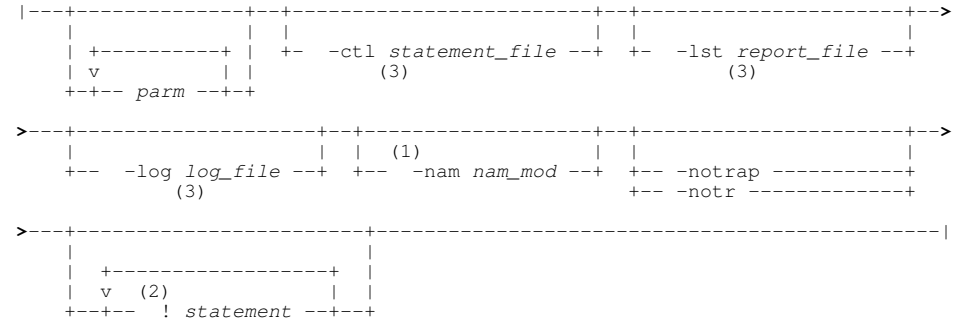
z/OS TSO (Alternative):



z/OS Batch:



Program Parameters:



(1) Program parameter -nam is applicable only in z/OS systems.

(2) Exclamation mark (!) is the default statement separator character. This may be changed by setting the SEP program environment option in SELCNAM.

(3) In Microsoft Windows, apostrophes (') are valid fileid characters. However, SELCOPY will strip apostrophes used to enclose a fileid.

Program Parameter Descriptions:

`-v`

Outputs 2 report lines to **stderr** which, for z/VM CMS, z/OS TSO and Batch corresponds to FILEDEF/DD name SYSOUT.

Note that **stderr** output may be redirected using the -log program parameter or a redirection symbol, 2> (two greater than) or to append to existing stderr output, 2>>.

The first is the standard output list header comprising the SELCOPY version & release, licensed organisation & location and current date & time. The second report line comprises the build level & date of the SELCOPY release.

```
SELCOPY/WNT 3.20 at CBL - Bridgend UK (Internal Only)        2013/12/10 17:30
          Build Level=001 2013/10/19 22:09 (Latest change).
```

`parm`

Identifies blank delimited parameter strings that are to be passed as input and processed by the SELCOPY control statements.

Up to 9 parameters may be specified and referenced within the SELCOPY program by parameter variables and the internal variable, ARG. If more than 9 parameter strings are specified, the program may access the additional parameters via the internal variable PARM.

Parameters must be enclosed in quotation marks (") or apostrophes (') if the parameter string contains any of the following:

1. Blank characters.
2. Characters that have special significance to the executing operating system.

   Beware that, in Unix systems, some special characters are interpreted despite being supplied in enclosing quotes, e.g. backslash (\), dollar ($), quotation mark (") and grave accent (`). To prevent this, the special character must be escaped using the escape character backslash (\).
3. Lower case alpha characters that must not be upper cased.

Note that some Unix operating systems (e.g. IBM AIX and Oracle Solaris) strip a single set of enclosing quotes (quotation marks or apostrophes) from each parameter specified via a command shell. Where this occurs, to preserve alpha character case in command line input, the parameter strings should be enclosed within two sets of quotes, one set of apostrophes, one set of quotation marks. e.g.

```
selcopy    '"./inp.test.file"'  '"/tmp/output.test.file"'

selcopy    "'Wuthering Heights'"  "'Emily Bronte''s Novel'"
```

`-ctl` *`statement_file`*
Identifies *statement_file*, the fileid of the SELCOPY program control statement file.

For Windows and POSIX systems, the directories specified by environment variable, PATH, will be searched if *statement_file* is a relative file path.

If -ctl *statement_file* is omitted, then SELCOPY accepts control statement input from **stdin** which, for z/VM CMS, z/OS TSO and Batch, corresponds to FILEDEF/DD name SYSIN. Control statements may also be provided as program parameters (see !*statement* below). These control statements will be processed before statements passed via *statement_file* or stdin input.

SELCOPY input via stdin is achieved using the redirection symbol less than (<) or as piped output from another source using the pipe symbol (|). If stdin is used as control statement input, it cannot then be used as data input via a READ STDIN operation. Therefore, specification of a control statement file via the -ctl parameter is recommended whenever possible. See your operating system documentation for use of stdin redirection and pipes.

If no control statements are passed to SELCOPY from any source, then, in a z/OS batch environment, the job ends with ERROR 524. In all other environments SELCOPY waits for input from the user's terminal and does not begin execution until the control statement input is ended. Control statement input via the terminal must be ended using the end of program indicator (composite symbol "/*" in the first column of the input) or, if no CARD input is performed, using the END operation keyword.

`-lst` *`report_file`*
Identifies *report_file*, the fileid of the SELCOPY list (diagnostic report and printed data) output file. Printed data is output from the PRINT and PLOG operations.

In Windows and POSIX (e.g. Linux and Unix) environments, SELCOPY will direct its list output in the following order of precedence:

1. The *report_file* fileid specified on -lst program parameter.
2. The **stdout** data stream if redirection symbol greater than (> or >>) is specified.
3. A fileid identified by the SLCLST environment variable.
4. The fileid "SELC.LST" in the current directory.

If stdout is used for list output, then it cannot then be used as data output via a WRITE STDOUT operation.

In z/VM CMS and z/OS non-POSIX environments (both TSO and Batch), SELCOPY will direct its list output in the following order of precedence:

1. The *report_file* fileid specified on -lst program parameter.
2. FILEDEF or DD name allocated to SYSPRINT.
3. For z/OS only, data set with DSN "*prefix*.SELC.LST" where *prefix* is the ACF user name assigned to the executing job.
4. For z/VM CMS only, the fileid "SELC LST A".

`-log` *`log_file`*
Identifies *log_file* as the destination file for all SELCOPY error messages that would otherwise be written to the terminal, and all data logged by the SELCOPY program using the LOG or PLOG operation.

If -log *log_file* is omitted, then SELCOPY directs logged output to **stderr** which, for z/VM CMS, z/OS TSO and Batch corresponds to FILEDEF/DD name SYSOUT. The -log program parameter is equivalent to **stderr** redirection using composite symbols two greater than (2> or 2>>).

`-nam` *`nam_mod`*
Applicable only in z/OS systems, identifies *nam_mod*, the alternate name of the CBLNAME options load module to be loaded by SELCOPY.

If it exists, the module named CBLNAME is loaded first to establish any update to the default statement separator character, otherwise the default of exclamation mark (!) is assumed. The alternate load module *nam_mod* is then loaded to establish all other options applicable to execution of SELCOPY.

`-notrap | -notr`
Disable default interrupt handling by SELCOPY so that any abnormal termination event, including interruption by the user, will be handled by the system.

This behaviour may also be achieved using the program environment option ABTRAP or TRAP.

`!statement`
Specifies one or more SELCOPY control statements that are to be processed **before** control statements input from another source. If specified, *statement* program parameters must follow any other program parameter. e.g.

```
SELCOPY  -ctl selccomp.ctl   !equ IN1 abc.fil   !equ IN2 xyz.fil
```

Each *statement* must be preceded by the statement separator character, which, by default, is exclamation mark (!) but may be set to another character using the program environment option, SEP, in the SELCNAM initialisation file. Note that, in the Unix csh or tcsh shell, exclamation mark has special significance and so must be escaped (/!) or followed by blank character.

If control statement input is not terminated by composite symbol "/*" as the first characters following a statement separator character (i.e. !/*), then SELCOPY will expect input of further control statements from another source. This may be the *statement_file* specified by program parameter -ctl or stdin input via file redirection, a pipe or the user's terminal.

If control statement input is terminated by a program parameter *statement*, then no attempt is made by SELCOPY to input further control statements as program parameters or from any other input (-ctl *statement_file* or stdin).

Sample Execution

The following provides samples of SELCOPY (SLC) execution in different environments:

z/OS JCL

Execution of SELCOPY requires specification of a control statement source and an output listing destination. In a z/OS (and z/VM CMS) environment, these default to the **SYSIN** and **SYSPRINT** ddname allocations, respectively.

Any input and output data set ddnames referenced within the SELCOPY control statements must also be allocated. e.g.

```
       <---+----1----+----2----+----3----+----4----+----5----+----6----+----7-
00001 //SAMPLE1 EXEC PGM=SLC
00002 //INPDD    DD  DISP=SHR,DSN=USER123.SLC.INPFILE
00003 //OUTDD    DD  DISP=SHR,DSN=USER123.SLC.OUTFILE
00004 //SYSPRINT DD  SYSOUT=*
00005 //SYSIN    DD  *
00006   READ   INPDD
00007   WRITE  OUTDD
00008 /*
```

The SYSIN control statement input and SYSPRINT list output may be overridden using parameters **-CTL** and **-LST** respectively. In batch, these parameters are passed via the JCL EXEC statement PARM field. Each of the -CTL and -LST parameters may specify a ddname or data set name (DSN). e.g.

```
       <---+----1----+----2----+----3----+----4----+----5----+----6----+----7-
00001 //SAMPLE2 EXEC PGM=SLC,PARM=('-CTL SLCINP -LST SLCPRT')
00002 //INPDD    DD  DISP=SHR,DSN=USER123.SLC.INPFILE
00003 //OUTDD    DD  DISP=SHR,DSN=USER123.SLC.OUTFILE
00004 //SLCPRT   DD  SYSOUT=*
00005 //SLCIN    DD  *
00006   READ   INPDD
00007   WRITE  OUTDD
00008 /*
```

Use of -CTL and -LST allows SYSIN and SYSPRINT ddnames to be used by other programs executed by the SELCOPY program via the CALL operation.

Existing input and output data sets used in the SELCOPY program need not be pre-allocated if parameter DSN is used on the SELCOPY I/O operations. In the following sample, SELCOPY will perform a dynamic allocation for input ddname INPDD and output ddname OUTDD for the specified data set names.

```
       <---+----1----+----2----+----3----+----4----+----5----+----6----+----7-
00001 //SAMPLE3 EXEC PGM=SLC,PARM=('-CTL SLCINP -LST SLCPRT')
00002 //SLCPRT   DD  SYSOUT=*
00003 //SLCIN    DD  *
00004   READ   INPDD   DSN='USER123.SLC.INPFILE'
00005   WRITE  OUTDD   DSN='USER123.SLC.OUTFILE'
00006 /*
```

Since SELCOPY also accepts control statement input via the parameter string, use of -CTL or SYSIN may be omitted altogether. This is not typical and can only be performed for brief SELCOPY programs since the input PARM field is restricted to 100 characters only.

```
      <---+----1----+----2----+----3----+----4----+----5----+----6----+----7-
00001 //SAMPLE4 EXEC PGM=SLC,PARM=('-LST SLCPRT !READ INPDD DSN="USER123.SLC.
00002 //             INPFILE" !WRITE OUTDD DSN="USER123.SLC.OUTFILE" !END')
00003 //SLCPRT   DD  SYSOUT=*
```

z/OS TSO/E

Like z/OS batch, SELCOPY execution in a z/OS TSO/E (or z/VM CMS) environment defaults control statement input and listing output to be SYSIN and SYSPRINT respectively. However, when running SELCOPY from a CLIST or REXX procedure, use of the -CTL and -LST parameters is preferrable since it removes the requirement of having to allocate and subsequently free SYSIN and SYSPRINT ddnames.

The following sample REXX procedure accepts a DSN or library member input parameter and executes SELCOPY with the -CTL and -LST parameter arguments assigned to values based on a standard library structure.

```
      <---+----1----+----2----+----3----+----4----+----5----+----6----+----7-
00001 /* REXX */
00002   parse arg inctl
00003
00004   address TSO
00005   pref  = sysvar("syspref")                /* TSO Prefix */
00006
00007   if pos(".",inctl) = 0                    /* No dot => member name. */
00008     then do; olist = pref".SLC.LISTING("inctl")"
00009              inctl = pref".SLC.CTL("inctl")"
00010          end
00011     else     olist = pref".SLC.LISTING(SLCLIST)"
00012
00013   "SLC    -CTL "inctl"    -LST "olist
```

SELCOPYi

SELCOPYi interactive utility component of the SELCOPY Product Suite for z/OS and z/VM CMS includes the facility **RUNSLC** for foreground execution of the SLC program module.

Control statements are read from the focus window text edit view (with or without unsaved alterations) and the resultant list output displayed in a new text edit view. The list output is assigned a temporary DSN but not saved.

Execution of primary command RUNSLC at the command prompt of a text edit view will start the facility. e.g.

```
Command> RUNSLC                                                Scroll> Csr
      <---+----1----+----2----+----3----+----4----+----5----+----6----+----7-
00001 ** NBJ.CTL(SSTEMP) ***             L=001 --- 2016/02/22 11:25:33
00002
00003    read indd   dsn='CBL.CBLI320.RS32002.DB2.DBRMLIB'   dir
00004    print
```

SELCOPYi also supports execution of programs in the TSO environment directly from any command prompt. Therefore, the SLC program may be executed at any SELCOPYi command prompt.

Furthermore, in combination with the Action Key facility supported by SELCOPYi text edit views, SLC commands may be saved in text files for subsequent execution. The following sample command text may be entered in any data set or library member (e.g. as comment data in a JCL batch job member). When edited using SELCOPYi, a command may be executed by positioning the cursor on the text and pressing the Action key (default <F4>).

```
<TSO   SLC   -CTL CBL.SELCOPY.CTL(SQ12836)   -LST CBL.SELCOPY.SYSPRINT(SQ12836)

<TSO   SLC   -LST *  !read 'USER123.SELCOPYI.CMX'  nordw  !print stopaft=22  !end
```

UNIX/Linux Shell

The command `selcopy` may be executed at any shell prompt or from a shell script.

By default, SELCOPY will accept control statement input from the stdin stream and direct its list output to stdout. Use of parameters `-ctl` and `-lst` will nominate the fileids to be used in place of stdin and stdout respectively. e.g.

```
selcopy  -ctl ~/selcopy/ctl/ssfstr     -lst ~/selcopy/lst/ssfstr
```

Using `-ctl` and `-lst` will allow data streamed from stdin and to stdout to be used on SELCOPY I/O operations.

If no input control file is specified, then SELCOPY waits for input from the default stdin input (i.e. the users terminal). If no list output file is specified, output is written to the file specified by environment variable **SLCLST** or SELC.LST if $SLCLST is null.

The following example assigns environment variable SLCLST which is used by the SELCOPY execution that follows. The SELCOPY input uses abbreviated operation and parameter keywords to reduce the length of the command syntax. It also

contains special characters "!" (exclamation mark) to separate the control statements and "'" (apostrophe) to delimit quoted character constants. Since these characters have significance in the command shell, they must be escaped using the shell escape character "\" (backslash) in order to avoid interpretation before being passed to SELCOPY.

The program itself will list all files in the user's home directory, input text from these files and report all lines containing the character string "user".

```
export SLCLST=~/selcopy/sstemp_lst
selcopy  \!rd \'%HOME%/*\' dd  \!if dir \!t log \!t gg \!if p any = \'user\' \!t log \'Data: \' fr 1 \!e
```

Microsoft Windows Shell

The command `selcopy` may be executed at any Microsoft Windows command shell, Visual Basic or command shell script.

Like SELCOPY on UNIX platforms, control statement input is read from the stdin stream and list output is directed to stdout by default. Parameters `-ctl` and `-lst` will nominate the fileids to be used in place of stdin and stdout respectively.

Also, stdin input from a terminal is used if no input control file is specified, and stdout output is written to the fileid assigned to environment variable SLCLST (default SELC.LST) if no list output file is specified.

The following example demonstrates setting the current directory prior to executing a SELCOPY that references relative fileid paths.

```
cd /d  "c:\Documents and Settings\NBJ\selcopy\"
selcopy  -ctl ctl\ssinv21_ctl.txt      -lst lst\ssinv21_lst.txt
```

Program Environment

Every execution of a SELCOPY program is subject to environment options which govern the operation of the SELCOPY executable. e.g. Statement separator character interpretation and formatting of SELCOPY's list file (diagnostic report and printed data) output.

Program environment options may be specified for individual SELCOPY programs but system-wide defaults, affecting all SELCOPY programs, may also be configured by the systems programmer, usually as part of the SELCOPY product install process.

This section focuses on configuration of system-wide defaults using the SELCNAM initialisation file and, for mainframe platforms only, the CBLNAME options module.

CBLNAME

Applicable to both the SELCOPY and SLC executables on mainframe systems, the CBLNAME option module is assembled from a configured assembler language source file using the assembly software provided as part of the operating system.

The CBLNAME assembler source file comprises a single, CBL supplied macro call with parameters that define many, but not all of the SELCOPY environment options. The most important option is the name of the SELCNAM initialisation file (data set).

Since CBLNAME defines system wide options, any update will affect all SELCOPY programs and should be restricted to authorised users only.

For full information on each CBLNAME option and for direction in updating the CBLNAME module in a z/OS system, see the "*SELCOPY Product Suite Customisation Guide*".

To update and re-assemble CBLNAME in z/VM CMS, see the *"SELCOPY Product Suite Install Guide for VM/CMS and VM/VSE Systems"*.

CBLNAME is the default name of the options module. However, for z/OS systems only, an alternative name may be specified when SELCOPY is started using the -nam *nam_mod* program option.

CBLNAME SLC Options

The following CBLNAME options are recognised by the SLC executable. Other options in CBLNAME apply to other executable elements of the SELCOPY Product Suite (i.e. SELCOPY, SELCOPYi and CBLVCAT).

SNamDsn='*selcopy.nam*'
: Identifies *selcopy.nam*, the name of the default SELCOPY environment initialisation file (data set) that gets allocated to FILEDEF/DD name SELCNAM. See SELCNAM or SELCOPY.NAM for description of this file.

This name may be overridden by allocating FILEDEF/DD name SELCNAM to a different file (data set) name prior to execution of SELCOPY.

If SNamDsn is not set and SELCNAM has not already been allocated, then *selcopy.nam* defaults to SELCOPY NAM in z/VM CMS and *prefix*.SELCOPY.NAM in z/OS systems, where *prefix* is the ACF user name assigned to the executing job.

Separator='*char*'
Separator specifies the default statement separator character, *char*.

This option may subsequently be overridden by the program environment option SEP, specified in SELCNAM or within the SELCOPY program control statements. The default is exclamation mark (!).

SOpsMsg=0 | ALL | NOSEL | NOCTL | NONE
Determines which, if any, of SELCOPY error message classes are logged to the log output file or SYSOUT. Default SYSOUT is the z/VM CMS console, the z/OS TSO console or z/OS operator console.

The default value is 0 (or ALL) indicating that all messages are logged.

SBannerMsg=YES | NO
Determines whether or not on startup of SELCOPY, the banner message is logged to the log output file or SYSOUT.

This option may subsequently be overridden by the program environment option (NO)BANNER, specified in SELCNAM or within the SELCOPY program control statements.

The default value is NO indicating that the banner message is not logged.

SRDW=YES | NO
Determines whether or not input record data belonging to a z/OS RECFM=V data set includes the 4-byte Record Descriptor Word (RDW) prefix.

This option may subsequently be overridden by the program environment option (NO)RDW, specified in SELCNAM or within the SELCOPY program control statements.

The default value is YES indicating that the RDW is included.

SNotFoundMsg=YES | NO
Determines whether or not the following, 25 character data message is moved to the input data field following an unsuccessful direct read by key or record number.

```
--- KEY/REC NOT FOUND ---
```

If this input message is suppressed, the program will have to test for the appropriate return code from the READ operation.

The default value is YES indicating that the message data is returned.

SPrtStopAft=*intval*
Specifies *intval*, a positive integer value which is used as the default STOPAFT value applied to all PRINT or PLOG operations only. i.e. Each PRINT and PLOG operation within a SELCOPY program will be executed no more than *intval* number of times in the same run.

This option may subsequently be overridden by specification of keyword parameter STOPAFT on individual PRINT and PLOG operations within the SELCOPY program control statements.

The default value is 0 indicating no limit for PRINT and STOPAFT 50 for LOG.

SCPCmds=YES | NO
Applicable only to z/VM systems, SCPCmds determines whether or not CP command streams may be executed from SELCOPY program control statements using the CP operation.

The default value is YES indicating that CP commands may be executed.

SELCNAM (SELCOPY.NAM)

In z/OS and z/VM systems, the SELCOPY initialisation file is referenced as SELCNAM. On all other systems, it is referenced as SELCOPY.NAM, the actual fileid of the file processed by SELCOPY. For the purpose of this manual, the SELCOPY initialisation file will be referenced as SELCNAM.

In z/OS and z/VM systems, the SELCNAM file is identified as being the data set allocated to the FILEDEF/DD name SELCNAM. If SELCNAM is not already allocated, it gets automatically allocated to the data set name specified by CBLNAME option SNamDsn. If SNamDsn is unset, a DSN of "tsopfx.SELCOPY.NAM" or "SELCOPY.NAM" is used. In all other systems, the SELCOPY.NAM file must exist in the file search path.

SELCNAM is an editable text file and may include only comment text and SELCOPY OPTION operations that define program environment options. Because updates to SELCNAM environment options affect exceution of all SELCOPY programs, it should be protected against unauthorised read-write access.

If a SELCNAM record contains the end of program input composite symbol (/*) in column 1, then records that follow this indicator are not read and SELCOPY processing continues with program control statement input.

Update of SELCNAM in a z/OS environment should not be performed on the active SELCNAM data set due to an exclusive ENQ being placed on the data set by the system editor. Appropriate procedures for updating SELCNAM in z/OS and z/VM CMS systems are detailed in the "*SELCOPY Product Suite Customisation Guide*" and the "*SELCOPY Product Suite Install Guide for VM/CMS and VM/VSE Systems*" respectively. Similar care should be taken when updating SELCNAM in any system where exclusive file locking occurs when editing a file. This will result in unsuccessful execution SELCOPY programs due to failure to open the SELCNAM file for input.

SELCNAM SLC Options

See operation keyword OPTION for detailed information on each of the program environment options.

The following SELCNAM program environment options are mandatory and must reference valid entries as supplied by Compute (Bridgend) Ltd:

SITE	Specifies the licensed company name and geographical location.
RANGE	Specifies the licensed, operational date range.
PASS	Specifies the unique 8-byte password expressed in hexadecimal.

The following program environment options may also be included in SELCNAM:

ABTRAP **TRAP** | **NOTRAP**	Enable or disable SELCOPY's Interrupt handling.
BANNER | **NOBANNER** **BAN** | **NOBAN**	Enable or disable logging of SELCOPY's banner message to the log output file or stderr (SYSOUT) on startup.
CALLTYPE	Controls the CALL operation linkage for calling external routines.
CBLSQLOG **SQLLOG** **LOGSQL**	Identifies an output file used specifically to log calls to the ODBC driver manager.
CONTMAX	Extends the maximum length of a control statements.
DATAWIDTH	Controls the width of printed data in SELCOPY's list output.
DEFAULTFP **DEFFP** **DFLTFP**	Controls the format of a floating point field if not explicitly specified.
DEFDIR	Identifies the default current, hierarchical file system working directory.
DUMPALL	Enable or disable print of second and subsequent, duplicate TYPE=D (dump) format print lines.
DUMPENC	Identify the characters to be used to delimit the character representation of TYPE=D (dump) format printed data.
ENVFAIL	Specifies the action to be taken when environment variable or parameter variable substitution fails because the variable has no assigned value.
ENVVAR | **NOENVVAR**	Enable or disable environment variable or parameter variable substitution.
ERRLIM **ERRMAX**	Determines the maximum number of control statement errors that are reported before control statement analysis stops and the program is terminated.
FILL **PAD**	If a user work area buffer is defined (OPTION WORKLEN), then this option identifies the default character to be used to overwrite residual data. This is data that is left in the buffer when an input record is of a shorter length than that of the previous record read by the same READ operation.
KEYENC | **NOKEYENC**	Applicable to the SELCOPY for Windows keyboard input feature, KEYENC identifies the Window special key name delimiter characters. Alternatively, special key name processing may be disabled.

KEYENCERR | **NOKEYENCERR**	Applicable to the SELCOPY for Windows keyboard input feature, this option enables or disables RC=8 set when a Windows special key is not recognised by SELCOPY.
LIBNAME	Identifies the name of the default library shared object in which routines called by a SELCOPY program may be found in option CALLTYPE DIRECT is in effect.
MFC	Indicates that all processing of VSAM files in SELCOPY programs is to use the Micro Focus VSAM interface.
ODBCPASS **OPASS**	Specifies the default password supplied by SELCOPY programs for user authorisation when attempting to access protected objects (databases) via ODBC.
PAGEDEPTH	Controls the maximum number of lines per page in SELCOPY's list output.
PAGEWIDTH	Controls the width of page headers and TYPE=D (dump) format printed data in SELCOPY's list output.
PRINTABLE	Identifies hexadecimal code points of characters within the local code page that are to be treated as being printable by SELCOPY for PRINT and PLOG output.
PRTCTL | **NOPCTL**	Enable or disable output of SELCOPY program control statements in the diagnostic report portion of the SELCOPY list output.
NOPSUM **NOPTOT**	Disable output of SELCOPY program execution summary information in the diagnostic report portion of the SELCOPY list output.
NOPRINT **NOP**	Disable output of both the SELCOPY program control statements and the execution summary information (i.e. all diagnostic information) in the SELCOPY list output.
RDW | **NORDW**	Enable or disable inclusion of the 4-byte Record Descriptor Word (RDW) prefix in a record read from a RECFM=V, RECFM=V2 or RECFM=MFV file.
SEP	Identifies the statement separator character.
SORT | **NOSORT** **SORTDIR**	For hierarchical file systems only, this option identifies the sort order of input directory records, and so the order of input files, returned by a READ DIR or DIRDATA operation.
SSN	Identifies the ODBC Data Source Name assigned to the default data object (database) accessed by SELCOPY programs via ODBC.
SUB | **NOSUB** **SUBDIR**	For hierarchical file systems only, this option identifies the number of levels of nested sub-directories, belonging to the input directory, that are to be processed by a READ DIR or DIRDATA operation.
TABSIN	For RECFM=U input only, this option identifies the tab interval which indicates the position to which text following an input tab character (x'09') will be shifted.
UNPRINTABLE	Identifies hexadecimal code points of characters within the local code page that are to be treated as being unprintable by SELCOPY for PRINT and PLOG output.
USER	Specifies the default userid supplied by SELCOPY programs for user authorisation when attempting to access protected objects (databases) via ODBC.

SELCMSG (SELCOPY.MSG)

The text of all control statement analysis and selection time messages set by SELCOPY is obtained from a single plain text file.

In z/OS and z/VM systems, the SELCOPY message file is referenced as SELCMSG. On all other systems, it is referenced as SELCOPY.MSG, the actual fileid of the file processed by SELCOPY. For the purpose of this manual, the SELCOPY message file will be referenced as SELCMSG.

In z/OS and z/VM systems, the SELCMSG file is identified as being the data set allocated to the FILEDEF/DD name SELCMSG. If SELCMSG is not already allocated, it gets automatically allocated to the data set name specified by CBLNAME option SNamDsn except that the last 3 characters of the DSN are replaced by "MSG". If SNamDsn is unset, a DSN of "tsopfx.SELCOPY.MSG" or "SELCOPY.MSG" is used. In all other systems, the SELCOPY.MSG file must exist in the file search path.

If the SELCMSG file is missing or cannot be found, then the text of any message returned by SELCOPY will be: `Error Number not in selcopy.msg file`

Program Processing

A SELCOPY program is started using the SELCOPY executable. SELCOPY's processing comprises the following 5 steps:

1. Establish licence details and program environment.
2. Control statement analysis and interpretation.
3. Output file open.
4. Statement selection and execution.
5. End of job file close.

The steps are performed in the order specified and are described in this section.

Establish Environment

SELCOPY's initial step is to read the SELCNAM file to verify licence details and then to process other OPTION operations to establish the program execution environment.

For z/OS and z/VM CMS systems, reading the SELCNAM data set involves first loading the CBLNAME module to discover the location of the SELCNAM data set, and then dynamically allocating the data set to FILEDEF/DD name SELCNAM for subsequent file open. Certain environment options are also initialised from CBLNAME, prior to processing SELCNAM. See Program Environment for details.

If licence details provided via the SITE, RANGE and PASS options are not consistent with the values supplied by Compute (Bridgend), then the following messages are returned in the logged output and/or the SELCOPY list diagnostic report.

```
*** ERROR 153 ***  INVALID OPT IN "SELCOPY.NAM" FILE
```

```
*** ERROR 124 ***  CHECK EXPIRY DATE
```

In this case, control statement analysis is still performed for the SELCOPY program, however, no subsequent steps are performed due to the control statement errors.

If any other invalid statement exists in the SELCNAM file, the following message is logged but processing continues. Note that *nnn* is the record number within SELCNAM at which the error was detected.

```
Syntax err in selcopy.nam :nnn
```

Control Statement Analysis

Before SELCOPY executes the supplied control statements, they are first analysed to detect any syntax errors and to process program environment operations including options that override system defaults.

Control statement analysis is performed on each statement read sequentially from start to finish. This order is not affected by any internal sub-routine calls or GOTO operations.

Control statements are analysed one at a time and are each converted into an internal control block. Program environment operations, e.g. OPTION and EQU, are converted and then executed immediately and so do not affect control statements that have already been analysed. e.g. An equated symbol substitution will not occur for occurrences of the symbol in statements for which analysis processing has already occurred.

The following identify the processing performed during control statement analysis.

Selection Identifiers

All control statements that execute an operation or sub-operation, or perform a variable assignment are assigned a unique selection identification number. Exceptions to this are program environment operations (e.g. EQU, DECLARE, OPTION), the IF logic operation and the AND and OR logic sub-operations.

The selection identifier assigned to a statement is the next integer value in an ascending sequence of integers starting at 1.

The selection number is displayed next to the control statement in the SELCOPY list diagnostic report and is used to reference the statement in the SEL-ID column of the SELCOPY program execution summary information.

Variable Substitution

Identifiers within a control statement that correspond to substitution variable names are replaced by the assigned value before syntax validation of the statement is performed.

If a substitution variable has no equivalent assigned value, then the following occurs:

- For equated symbols, no substitution is performed and syntax validation on the statement is performed as is. Note that an equated symbol value will not exist in the case where a statement which references the equated symbol is processed before the statement containing its definition (EQU operation).
- For parameter variables and environment variables, the action defined by the ENVFAIL option.

File Open

Statements containing a READ or WRITE operation on a file with no explicit or implied DEFER parameter, will first perform any required dynamic allocation for the specified fileid (DSN) and then open the file for input, input for update or output processing as required by the operation. If DEFER is in effect, the open of the input or output file is deferred until the statement is executed at selection time processing.

Input files are opened immediately following successful validation of the statement containing the READ operation. Output files are opened at the end of control statement analysis, once all control statements have been processed with no resulting errors.

By default, SELCOPY WRITE output to non-VSAM files where parameter APPEND is omitted will re-initialise the file so that the file size is zero before writing new data. Therefore, delaying the open of output files until selection time processing is about to begin ensures that data that may already exist in the output files is preserved if the SELCOPY program fails to execute.

List input, read via a READ LIST operation, is generated whenever the open is executed. Declaration and open of table cursors via ODBC for READ and WRITE operations that are to be performed on data base tables, is always deferred until selection time processing. A WRITE operation that performs window keystroke output does not require an open.

Note that file open is deferred for a file if any of the following is true:

1. Parameter DEFER is specified on the first READ operation or any WRITE operation that specifies the file name.
2. Dynamic allocation is to be performed for the file name and the fileid on the operation is specified as a field definition or a declared variable.
3. For input, a statement containing an OPEN or CLOSE operation for the file name is processed before the statement containing the first READ operation for the same file.
   For output, a statement exists which contains an OPEN or CLOSE operation for the same file name specified by the WRITE operation. The OPEN or CLOSE statement may be processed after the WRITE statement.

Prime Input

During statement analysis, the prime input is determined as being the file or database result table specified by the first READ statement, regardless of whether it is opened at control statement analysis or the open is deferred until selection time processing.

The concept of a prime input is used by SELCOPY execution to determine the natural end of SELCOPY selection time processing. The file name assigned to the prime input is also the default used when a file name argument is not specified on an IF operation that tests for EOF, INCOUNT, DIR or DATA.

By default, SELCOPY selection time processing finishes and end of job processing is triggered when a READ operation is performed for the prime input and there are no more records left to read. e.g. the last record has already been read for forward sequential input.

If no READ operation statements exist then no prime input is established and the program ends following execution of the last selected control statement.

Control Statement Errors

Control statement analysis errors may be flagged for individual control statements that are determined to be invalid or, in the case of READ and WRITE operations, for which a file allocation or open has failed.

If an error is flagged for a control statement the following occurs:

1. If output of SELCOPY program control statements to the diagnostic report portion of the SELCOPY list output has been disabled (option NOPCTL or NOPRINT), then it is re-enabled. All subsequent statements processed by control statement analysis will be written to the SELCOPY list.
2. The control statement in error is written to the SELCOPY list.

3. If possible, the offending identifier, constant or operator element within the statement text is identified by an asterisk (*) in the SELCOPY list line immediately following the statement.

4. The relevant control statement error message is written to the SELCOPY list and, if the first error encountered, it is also written to the log output (stderr/SYSOUT). Note that output of error messages to log output on z/OS and z/VM platforms may be controlled by CBLNAME option SOpsMsg.

5. The control statement error count is incremented by 1.

6. SELCOPY return code 52 is set.

If the count of control statement errors reaches the maximum permitted, then control statement analysis terminates immediately and end of job processing begins. The error count maximum is determined by the ERRLIM (or ERRMAX) option which has a default setting of 10. Similarly, end of job processing is invoked following normal completion of control statement analysis if any errors have been flagged.

Individual control statement errors are described in Control Statement Analysis Error Messages.

Selection Time Processing

If no statements have been flagged as being in error and all necessary file allocation/open has been performed successfully, then selection time processing begins.

Selection time processing involves the systematic selection and execution of the SELCOPY statements which includes resolution of current values for variables and expressions specified as parameters to operations.

By default, control statements are selected in order of ascending selection identifier. This is common in only very simple programs that do not involve conditional operations or logic flow operations that perform a direct branch to a label statement. In most SELCOPY programs, the default order of control statement selection is influenced by logic flow operations (e.g. GOTO, DO) and the result of logical operations/sub-operations IF, AND, OR which govern the selection of conditional sub-operations THEN, ELSE.

Implied Loop

If a prime input exists, then, following execution of the last selected input control statement, selection processing loops to continue its processing from the first selectable control statement. This behaviour is equivalent to that of the GOTO GET operation. Note that the first selectable control statement may be a conditional operation subject to IF, AND, OR.

This processing continues until no further records are read from the prime input either because end of file (or result table) is flagged or because STOPAFT parameter thresholds have been satisfied for all prime input READ operations or all output (PRINT, LOG, WRITE) operations.

This loop processing may be controlled within the SELCOPY program using an IF/AND/OR EOF condition to test the end of prime input and so perform additional processing before executing the EOJ operation to force end-of-job. Similarly, GOTO EOJ may be executed on any statement to end the execution before all prime input records have been read.

Note that there is no implied loop if no prime input exists, in which case selection time processing ends after the last selected control statement has been executed.

Selection Time Errors

Selection time errors may be flagged for individual control statements that fail to execute properly.

Selection time errors are usually errors that cannot be established by SELCOPY's control statement analysis and only occur when an attempt to execute the statement is performed. e.g. Failure to dynamically allocate a file due to an invalid fileid having been specified in the DSN field definition.

If an error is flagged at selection time the following occurs:

1. The relevant selection time error message lines and any accompanying dump print are immediately written to the SELCOPY list. These message lines will appear before the diagnostics summary block.

   The message is prefixed by the selection identifier number of the statement on which the error occurred. e.g. `(SEL---12)` indicates the statement assigned the selection identifier number 12.

   If the error occurs on an IF, AND or OR operation, then the selection identifier used is that of the first THEN sub-operation statement that follows. Furthermore, the selection identifier is displayed with only 1 preceding hyphen (-). e.g. `(SEL  -12)` refers to the IF, AND or OR operation that occurs immediately before the THEN statement assigned the selection identifier number 12.

2. The same selection time error message lines are written to the SELCOPY log output (stderr/SYSOUT). The message is prefixed by the SELCOPY version and release. e.g. SELCOPY/WNT 3.20.

Additional messages may also have been written to the log by SELCOPY's internal functions for diagnostic purposes. e.g. The following message, logged by internal function cblfio, provides more accurate information as to the reason for the I/O error:

```
cblfio: WinErr=0161 The specified path is invalid.    (Input) F="fileid"
```

Note that output of error messages to log output on z/OS and z/VM platforms may be controlled by CBLNAME option SOpsMsg.

3. Selection time processing terminates immediately and end of job processing begins.

Individual selection time errors are described in Selection Time Error Messages.

End of Job Processing

Before the SELCOPY execution terminates, the following end of job tasks are performed:

1. Free storage obtained by SELCOPY during execution.
2. Free any dynamically allocated files.
3. Close all input and output files and close all ODBC connections.
4. For z/OS and z/VM only, drop any lists generated by READ LIST operations.
5. Write the diagnostic summary block to SELCOPY list.
6. Write any non-zero return code warning message to the SELCOPY list which may include the selection identifier of the first statement in error.
7. Write the standard SELCOPY report footer lines to SELCOPY list.
8. Close the SELCOPY list and SELCOPY log output.

SELCOPY List Output

By default, the SELCOPY program list output contains both diagnostic information and any text printed by the SELCOPY program using the PRINT or PLOG operations.

The destination of the SELCOPY list output is as described by the SELCOPY command -lst parameter in section *"Invoking the Executable"*.

On Windows and Unix systems, list output is of undefined record format (RECFM=U) where applicable print control characters, carriage return (x'0D'), line feed (X'0A') and form feed (x'0C') are used to control the formatting of text output.

On z/OS and z/VM systems, list output is of record format VBA which uses standard mainframe ASA print control characters to control the formatting of the text output. Therefore, when browsing or editing the list output generated on these systems, the first column of the display contains the ASA characters (1, 0 or b) and does not constitute part of the SELCOPY list output text. Likewise the list output page width, as defined by the PAGEWIDTH option, does not include the ASA column.

SELCOPY list output is organised in pages of specific page depth as defined by the PAGEDEPTH option. Each page has configurable header lines and, if not suppressed by option NOPRINT, the last page ends with standard footer lines.

The line number within the current page of the last line written to the list is maintained by internal variable, LINE. Where the value of LINE is equal to the current page depth value or if assigned a value lower than its current value, then the next output to the list will start a new page and the outputted text will be written immediately following the new page's header lines.

```
SELCOPY/WNT 3.20 at CBL - Bridgend UK (Internal Only)                          2014/01/24 15:20   PAGE   1
-------------------------------------------------------                        ----------------   --------


          option sortdir=n  subdir=0  datawidth=80  pagewidth=110

        ** c:\nbj\examples\dd01_ctl.txt *** L=002 --- 2014/01/24 15:20:00 (L05)
        *
        * SELCOPY to generate a log of activity on a specific FTP server.
        *  1. Read multiple syslog files whose fileids match a specified fileid mask.
        *     (Fileids are sorted by date descending, with sub directories excluded.)
        *  2. Limit the number of input records processed to be 5,000.
        *  3. At the start of each log file, write the fileid to the output file
        *     following a "***" marker.
        *  4. Write only records that contain the specified literal strings.


    1.    read   indd   dsn="c:\hst\log\syslog\2013*.rt1"   dirdata  stopaft 5000


          if dir                          ** File directory records.
    2.      then write outlog   dsn="c:\tmp\ftp_activity.txt"  from "*** "  from 60
    3.      then goto get                     * Return to top of main processing loop.

                                        ** File data records.

          if  pos any   = "dst=172.168.1.100"   * Sets @ pointer at the first match in the data.
          and pos @+58  = "dst_port=21"         * Test fixed offset from the @ pointer position.
          and pos 1, @  = "src="                * Sets @ pointer at source IP address.
    4.      then write outlog
    5.      then print  from @, @+79  stopaft 5 * Print text of interest.


   INPUT   SEL SEL                                                                                  RECORD
   RECNO   TOT ID.          1         2         3         4         5         6         7         8  LENGTH
   -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
      59     1   5 src=95.64.37.10 srcname=95.64.37.10 src_port=80 dst=172.168.1.100 dstname=172.16    695
     673     2   5 src=173.254.28.40 srcname=173.254.28.40 src_port=80 dst=172.168.1.100 dstname=17    699
     183     3   5 src=95.64.37.10 srcname=95.64.37.10 src_port=80 dst=172.168.1.100 dstname=172.16    695
    2825     4   5 src=161.69.13.21 srcname=161.69.13.21 src_port=443 dst=172.168.1.100 dstname=172    698
    2826     5   5 src=161.69.13.21 srcname=161.69.13.21 src_port=443 dst=172.168.1.100 dstname=172    698
                 ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1         5,000 READ INDD       2048  2046 U              5       C:\hst\log\syslog\2013*.rt1
                                  *EOF*NOT*REACHED*
    2             5 WR   OUTLOG     2048   713 U             13       C:\tmp\ftp_activity.txt
    3             5
    4             8 WR   OUTLOG     2048   713 U             13       C:\tmp\ftp_activity.txt
    5             5

***WARNING***                   4 = RETURN CODE FROM SELCOPY

    ** SELCOPY/WNT 3.20.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                    ** Expiry: 17 Jul 2014 **
```

Figure 1. Sample SELCOPY List Output.

SLCLST Environment Variable

For Windows, Unix and OS/400 operating systems, the environment variable SLCLST may be set to identify the default fileid to which SELCOPY list output is written when no -lst program option is specified on the SELCOPY command. Note that Unix systems, for which names are case sensitive, must assign the variable name in upper case.

The fileid assigned to SLCLST will override the SELCOPY default, defined as being fileid "SELC.LST" in the current working directory. Sample commands for setting environment variable SLCLST in each supported system follow:

Windows command shell

```
set     SLCLST=c:\tmp\s.lst
```

Unix Bourne or Korn shells (sh, bash or ksh)

```
export  SLCLST=/home/nbj/selc_list
```

Unix C shell (csh or tcsh)

```
setenv  SLCLST /home/nbj/selcopy_list
```

OS/400

```
ADDENVVAR ENVVAR(SLCLST) VALUE('/nbj/slst')
```

Header Lines

Each page of the SELCOPY List output begins with 3 header lines:

1. The header text which includes a title in addition to the current date, time and page number.
2. A line containing only hyphen/minus symbols (-) that underline text in the first header line.
3. A blank line.

The existence of these lines means that the value assigned to internal variable LINE will never be less than 3.

The header title is left adjusted within the first header line so that it occupies the first printable character. Note that ASA characters occupy column 1 for z/OS and z/VM CMS list output. The remaining group of header line items (current date, time and page number) are right adjusted at the column value specified by option PAGEWIDTH.

Note that the width of the page number value is 4 characters. If >9999 pages are written to the SELCOPY list output, the page numbers of pages following page 9999 are displayed as 4 asterisks (****).

The default SELCOPY header title is as follows:

```
SELCOPY/xxx n.nn at site_name
```

Where:

- *xxx* is a 3 character abbreviation representing the operating system as defined by generic terms in *"Documentation Notes."* e.g. SELCOPY/WNT implies SELCOPY for Windows.
- *n.nn* is the SELCOPY release. e.g. 3.20
- *site_name* is the company name and location as defined by the SITE option in the SELCNAM (SELCOPY.NAM) file. e.g. Compute (Bridgend) Ltd - Bridgend UK

This default may be overridden during control statement analysis using the HEAD option or the HEAD parameter of the REPORT operation. During selection time processing, the header title may be updated using internal variable, HEAD.

Note that, because lines are written to the SELCOPY list output following analysis of the first control statement, then, unless the first statement sets the header title, then the title displayed on the first output page header will be the SELCOPY default.

If any control statement performs a TYPE=S (System) print operation, then SELCOPY list page formatting, including output of page headers, is suspended until end of job processing is started.

Control Statements

Unless options NOPRINT or NOPCTL are set, control statements are written one at a time to the SELCOPY list output as they are processed by control statement analysis.

If a selection identifier number has been assigned, it appears right-justified with a trailing period (.) in the first 7 characters of the output record containing the control statement. Otherwise, the first 7 characters contain blanks. The text of the control statement itself is indented so that it starts in column 10 of the list output record.

If desired, OPTION operation statements with keywords PRTCTL or NOPCTL may be interspersed throughout a SELCOPY program to determine whether or not the control statements that follow the OPTION statement are included in the diagnostic output. PRTCTL switches on control statement output, NOPCTL switches it off. By default, PRTCTL is in effect and so all statements of the control statement input source, including comment text and blank lines, are written to the SELCOPY list.

If a control statement analysis error control statement analysis error occurs, then control statement printing is reactivated if NOPRINT/NOPCTL is in effect and the appropriate error highlighting and message text is written immediately following the statement in error. e.g.

```
      2.    PRINT   This will give a control statement error.
                    ****
*** ERROR 042 ***  I/O LIT MUST BE IN QUOTES / FROM MISSING
```

The maximum number of control statement analysis errors reported in the SELCOPY list is determined by the ERRLIM option value (default 10).

A statement that follows the statement separation character (!) is processed after the statement that occurs immediately before it. Consequently, the separator character is removed and the statement that follows is written to a new line of the list output but further indented so that it occurs at the same offset as it does within the input record. e.g.

```
read   indd  !print     * Print input records.
```

The SELCOPY list output:

```
    1.    read   indd
    2.                   print     * Print input records.
```

Statements that span more than one input record using the statement continuation character (\) occupy the same number of lines in the SELCOPY list output. The continuation character itself is preserved in the statement list output.

For the purpose of improved readability, additional blank lines are written to the list output. e.g. A blank line is written before each statement containing an IF logical operation and before the statement assigned the first selection identifier, and two blank lines are written after the last control statement processed.

Print Block

Any PRINT or PLOG operation executed during selection time processing will write text to the SELCOPY list output following the control statement diagnostic output but before the summary block.

The type of printed output, as specified by the TYPE parameter, determines the format of the data written to the list. For all SELCOPY print output, except TYPE=S (System), the first execution of a PRINT or PLOG operation for the current list page will write the 3 standard print block header lines as follows:

```
INPUT   SEL SEL                                                  RECORD
RECNO   TOT ID.          1         2         3         4         5 LENGTH
-----   --- --- ....,....0....,....0....,....0....,....0....,....0 ------
```

When no further text is to be printed or before starting a new list page, the standard print block footer line is written. This comprises a scale of length equal to the datawidth, located immediately under the printed text. This scale contains dots/periods (.) for each character position that is not a multiple of 5, a comma (,) for character positions that are a multiple of 5 but not a multiple of 10, and numbers 0 to 9 representing character positions at an interval of 10.

```
....,....1....,....2....,....3....,....4....,....5
```

Unless otherwise stated, numeric values reported in the print block which overflow the allotted column field width will be reported as a number of asterisk symbols (*) equal to the width of the column field.

INPUT RECNO
Identifies the record or row number of the prime input at the time the print operation occurred. If the prime input has more than one input source (i.e. READ with a CAT sub-operation or the prime input gets re-opened with a different *fileid*) the record number is reset for each input source. Note that a reset does not occur where a prime input is identified by a z/OS ddname which is allocated to a concatenation of data sets.

The INPUT RECNO column starts in record position 1, has a width of 9 characters and both the column header text and column values are right adjusted within the column area.

SEL TOT
Identifies the execution occurrence of the print operation that generated this line of printed text. e.g. a value of 4 would indicate the 4th execution of this particular PRINT or PLOG operation within the current program execution. A count is maintained for each occurrence of a PRINT/PLOG selection within the SELCOPY control statements.

The SEL TOT column starts in record position 10, has a width of 6 characters and both the column header text and column values are right adjusted within the column area.

SEL ID.
The selection identifier number assigned to the PRINT or PLOG operation that produced the line of printed output.

The SEL ID. column starts in record position 16, has a width of 4 characters and both the column header text and column values are right adjusted within the column area.

Scale
Applicable only to print output other than TYPE=D (Dump) and TYPE=S (System), a decimal scale counting guide is included as a header, below which the printed data is displayed for the selected PRINT or PLOG operation. The width of the scale line dictates the length of printed text to be displayed on each line of the print block.

More than one print line may be written to represent the same length of printed text depending on the type of printed output specified. e.g. In addition to the character text print, PRINT TYPE=B will include an extra 2 lines that display an up-down hexadecimal representation of the printed text as well as an intervening blank line.

The scale may occupy all 3 of the print block header lines, one each for the hundreds, tens and units displayed at 10 character intervals. The third header line contains a line of dots/periods (.) for each character position that is not a multiple of 5, a comma (,) for character positions that are a multiple of 5 but not a multiple of 10, and a zero (0) for character positions that are a multiple of 10.

The column scale text and printed text start in record position 21. The width of this column is governed by the value assigned by the DATAWIDTH option.

If the printed text is longer than the DATAWIDTH value, the text continues on second and subsequent print lines. Each of these continued lines of printed text are prefixed by a value representing the offset of the text from the start of the printed field. This value starts in record position 6, has a width of 8 characters and is of the format *+nnn,nnn*, where non-significant leading zeros and commas are suppressed and the plus symbol (+) is right adjusted to the first significant numeric. Note that, for groups of print lines that represent the same printed text (e.g. PRINT TYPE=B), the offset value is displayed before the first print line of the group only.

For example, with OPTION DATAWIDTH=50 in effect, a printed prime input record of length 215 is displayed as:

```
INPUT   SEL SEL                                                  RECORD
RECNO   TOT ID.          1         2         3         4         5 LENGTH
-----   --- --- ....,....0...,....0...,....0...,....0...,....0 ------
    9     9   7 10010005....Mrs...Patricia.......Sample...........    215
      +50       ...Mrs Sample.............xxxxxxxx xxxxx.........
     +100       .....xx. xxxxxxxxx xxx.............................
     +150       ..........Fxxxxxxxxx......Kxxx..xxxx xxx..xxxxxx x
     +200       xxxxxx......C..
                ....,....1...,....2...,....3...,....4...,....5
```

RECORD LENGTH

Applicable only to print output other than TYPE=D (Dump) and TYPE=S (System), this column displays the length of the last record or row read from the prime input object at the time the print operation was executed.

The RECORD LENGTH column starts in record position 21 plus the DATAWIDTH value and has a width of 7 characters. The column header text is right adjusted at one column beyond the width of the column, whereas the column values are right adjusted within the column area width.

Display of this column may be suppressed using the PRRECLEN=NO option.

PRINT Block - TYPE=D Output

Printed text that is in TYPE=D (Dump) format also writes 3 header lines but includes only the INPUT RECNO, SEL TOT and SEL ID. columns. (i.e. No scale or RECORD LENGTH columns.)

```
INPUT   SEL SEL
RECNO   TOT ID.
-----   --- ---
    1     1   2            71
0000  2A2A2063 3A5C6E62 6A5C6361 5C4C3035     |** c:\nbj\ca\L05|
0010  2E636D78 202A2A2A 20202020 20202020     |.cmx ***        |
0020  2020204C 3D383130 202D2D2D 20323031     |   L=810 --- 201|
0030  352F3035 2F323820 31313A35 373A3132     |5/05/28 11:57:12|
0040  2020284C 303529                         |  (L05)         |
      2     2   2             0
      3     3   2            58
0000  3C736574 70742037 3220602A 20216363     |<setpt 72 `* !cc|
0010  20636020 20202020 20202020 20202020     | c`             |
0020  20207C20 53657420 706F696E 74732061     |  | Set points a|
0030  6E642063 6F6C6F75 722E                  |nd colour.      |
```

Although no RECORD LENGTH column header is displayed, the length of the last record or row read from the prime input at the time the print operation was executed, is displayed to the right of the SEL ID column value. The value is right-adjusted in a column starting at record position 29 for a width of 5 characters.

The printed text is written on the lines following the statistical values and spans the entire width of the list output starting at record position 3. The print format is the standard IBM system dump display where each printed line contains a hexadecimal value offset from the start of the printed text, followed by the hexadecimal and character representation of the text at that offset.

Offset

The offset value is a 4 digit hexadecimal (hex) numeric value, printed so that the hex digits are displayed side-by-side and occupying 4 characters of the print line. Therefore, an offset may be a value in the range x'0000' to x'FFFF' (decimal 65535). For offset values above this maximum, the displayed value is truncated on the left so that high order digits are not displayed.

Hexadecimal Display

Hexadecimal (hex) display of the printed text is such that each character is represented by 2 hex digits in the range x'00' to x'FF' as defined by EBCDIC or ASCII character encoding standards. These hex digits are displayed side-by-side in a dump print. i.e. Each character of printed text occupies 2 characters of printed hexadecimal representation. For convenience, the hex display is grouped into 4 characters of printed text (8 hex digits) with a single intervening blank.

Character Display

Character representation of the printed text follows the last character of the hexadecimal display with 5 intervening blanks. The character display is enclosed within special characters as defined by the DUMPENC option - default or-symbol (|).

The first line of printed text starts at offset zero (0000) with subsequent lines containing text at consecutive offsets up to the end of the printed field.

The length of printed text displayed on each line of the TYPE=D output is always a multiple of 4 (1 fullword) and is governed by the prevailing value for page width (option PAGEWIDTH) as detailed below.

PAGEWIDTH values	Printed Text Length		PAGEWIDTH values	Printed Text Length
66	12 **(3 fullwords)**		93-105	24 **(6 fullwords)**
67-79	16 **(4 fullwords)**		106-118	28 **(7 fullwords)**
80-92	20 **(5 fullwords)**		119-156	32 **(8 fullwords)**

If option DUMPALL=NO is in effect, consecutive TYPE=D print lines that contain the same text, will be grouped together and condensed so that only the first line of the group is displayed with all remaining lines in the group replaced with a single print line. This single line contains the offset of the 2nd line in the group, the literal "=same=" followed by the number of lines represented by this print line. The character text area of this line is all blank and enclosed by the alternate DUMPENC option character - default colon (:).

```
  INPUT   SEL SEL
  RECNO   TOT ID.
  -----   --- ---
      0     1  11           190
0000  206C696E 65732073 75707072 65737365     | lines suppresse|
0010  642E2020 20202020 20202020 20202020     |d.              |
0020  20202020 20202020 20202020 20202020     |                |
0030       =same=          (7 lines)          :                :
00A0  20202020 20202020 44696666 20646174     |        Diff dat|
00B0  612E2020 20202020 20202020 2020         |a.              |
```

Summary (Totals) Block

Unless options NOPRINT or PRTSUM=0 (synonyms NOPTOT, NOPSUM) are set, the summary of control statements execution is written to the SELCOPY list output during end of job processing.

The summary block reports the number of times each control statement was executed as well as information on each input and output object. The level of detail and, hence, the amount of report lines displayed in the summary block, is determined by the PRTSUM option value (default 2).

Unless otherwise stated, numeric values reported in the summary block that overflow the allotted column field width will be reported as a number of asterisk symbols (*) equal to the width of the column field.

The 3 standard summary block header lines are as follow:

```
SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
```

Starting in record position 1, **SUMMARY..** occupies the first header line and identifies the start of the summary diagnostic information block. The remaining 2 header lines identify column names below which statistical values are displayed for each individual statement or group of statements eligible for execution during selection time processing.

Information displayed for I/O operations may occupy multiple lines of the summary block report and, depending on the PRTSUM level, additional lines may be included for labels and operations not assigned a selection identifier number. Further lines may be included to report SELCOPY warning messages which immediately following the statement selection report lines to which they apply. See Summary Block Messages for detailed descriptions of these messages.

Named summary block columns SEL-ID and SELTOT contain values for all statements for which a selection identifier number has been assigned. The remaining columns potentially contain values for I/O operations CAT, CLOSE, DELETE, INSERT, OPEN, READ, UPDATE and WRITE only.

Summary Block Selection Statistics Columns

SEL-ID

Specifies the selection identifier number or range of selection identifier numbers corresponding to one or more executable statements to which summary details in the current report line are applicable.

Consecutive selection identifiers are grouped together if the statements to which they are assigned have been executed the same total number of times (see the SELTOT column). A statement identifier is excluded from this grouping when the statement to which it is assigned contains a summary block comment (starting with compound symbol *>) or is one of the operations: CAT, CLOSE, DELETE, INSERT, ODBC, OPEN, READ, UPDATE or WRITE.

Similarly, a group of selection identifiers will be split into smaller groups where option PRTSUM=2 is in effect and a programmer-defined label occurs within the group, or grouping suppressed altogether when option PRTSUM=3 is in effect.

The SEL-ID column header starts in record position 2.

The first (or only) selection identifier number is right adjusted in an area starting at record position 1 of width 5 characters. If a range of selection identifiers are represented, this number is the first identifier in the range. The second identifier number represents the last identifier in the range and is right adjusted in an area starting at record position 6 of width 5 characters. Non-significant zeroes in the first number are replaced with blanks whereas non-significant zeroes in the second number are replaced with hyphen (-) symbols.

```
SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1         5,319 READ INCMX      2048   211 U           5,319       C:\nbj\ca\xxx.cmx
    2---13      320
   14            22  Summary block comment.
```

SELTOT
Specifies the total number of times the statement or group of statements, represented by selection identifier(s) displayed in the SEL-ID column, have been executed in the SELCOPY run.

The SELTOT column header starts in record position 14.

The totals number is displayed with comma (,) punctuation after every 3 digits starting from the units digit and is right adjusted in an area starting at record position 7 of width 13 characters.

This area overlaps the area occupied by the second selection identifier number in a range group of identifiers displayed in SEL-ID. However, this has no destructive affect unless the total number of executions exceeds 9,999,999, in which case the second identifier number in the range of selection identifiers is suppressed.

If the selection total value exceeds 2,147,483,647 (hex 7FFF,FFF), then `+2.1G+` is displayed instead.

opword
This unheaded column occupies the area between the SELTOT and FILE columns. The contents of this column are dependent on the level of diagnostic information in effect (PRTSUM) and can potentially span all subsequent columns.

PRTSUM=1
The column starts in record position 21 and has a width of 4 characters.

It contains one of the I/O operation words (READ, CAT, WRITE, INSERT, UPDATE or DELETE) as executed by the statement identified by the selection identifier in SEL-ID. Note that summary block comments on these types of statement are ignored.

For all other statements or groups of statements assigned identifiers, this column contains blanks unless a summary block comment has been specified or a summary block message has been returned. If either of these conditions is true, the statement for which the comment or message text is to be displayed, is not included in a statement group but is reported separately. Comment text starts at offset 1 of this column, message text at various offsets, with text spanning all subsequent columns.

If the summary block report line contains comment text or I/O operation data, then a summary block message is reported on a line of its own, immediately following the statement report line to which it applies.

PRTSUM=2
In addition to the content described for PRTSUM=1, this column contains the names of programmer defined labels and any unconditional RETURN operations. For highlighting purposes, the label names are displayed with enclosing equals (=) symbols and a preceding blank line, whereas RETURN operations displayed as `=ret=`.

PRTSUM=3
The column starts in record position 21 and has a width of 10 characters.

With the exception of summary block comments, this column has the same content as for PRTSUM=1 and 2 but with the In addition to the content described for PRTSUM=1 and 2, this column contains an entry for each run-time executable statement, regardless of whether it has been assigned a selection identifier. For each of these statements, this column displays either the abbreviated name of the operation being executed or the name of the internal variable or @Variable to which a value is being assigned.

Operation names, other than those I/O operations itemised in PRTSUM=1 above, and names of internal variable/@Variable assignments may span summary block columns that follow.

Unlike PRTSUM=1 and 2, all comment text, including summary block comments, are aligned at a fixed record position. Comment text includes the preceding asterisk (*) symbol and overlays blank entries in subsequent columns of the same summary line. Where specified on one of the I/O operations, the subsequent column entries are not blank and so the comment text is aligned at the same, fixed record position but on a new line.

Comment text on programmer-defined labels is aligned at record position 53. On all other statements, comment text is aligned at record position 73.

Summary Block I/O Operation Columns

FILE
Identifies an up to 8 character file name assigned to the object on which the I/O operation is performed.

By default, the FILE column starts at record position 26, has a width of 8 characters and both the column header text and column values are left adjusted within the column area. If PRTSUM=3 is in effect, this column starts at record position 32.

BLKSIZE
Displays the block size used for the I/O operation. The block size defines the size of the buffer used by SELCOPY when performing data I/O on the specified object, and is the maximum size of a block for file objects defined with a blocked record format (RECFM). See LRECL column below.

For VSAM data sets processed in a z/OS or z/VM CMS environment, the reported BLKSIZE value is the maximum record size (RECORDSIZE) value defined for the cluster data records.

By default, the BLKSIZE column starts at record position 34, has a width of 7 characters and column values are right adjusted within the column area. The column header text is located at record position 35. If PRTSUM=3 is in effect, the column values start at record position 40 with header text starting at record position 41.

LRECL

Displays values for the input or output object in the following order:

1. The length of the longest input or output record processed for the specified object.

   This column value is numerical and starts at record position 42, has a width of 5 characters and is right adjusted within the column value area. If PRTSUM=3 is in effect, the column value starts at record position 48.

2. The format (RECFM) of the records processed.

   This column contains a character value and starts at record position 48, has a width of 3 characters and is right adjusted within the column value area. If PRTSUM=3 is in effect, the column value starts at record position 56.

The column header text is located at record position 44 or, if PRTSUM=3 is in effect, position 50.

The basic record formats displayed in the second column are:

F	Fixed length format records.
V	Variable length format records.
U	Undefined length format records.

Unless a RECFM value is specified as a parameter on the SELCOPY I/O operation or the RECFM value can be established from another source (e.g. z/OS VTOC or JFCB), then the default input record format is U (undefined) and the default output record format is the same as the prime input.

For native z/OS data sets and z/VM CMS files, the record format displayed in this column will be the RECFM defined when the file (data set) was allocated/created. See operating system documentation for possible RECFM values and their definitions. e.g. For z/OS, see *"MVS JCL Reference"*.)

VSAM RRDS input/output, ODBC based input/output and SELCOPY list input are all reported as being record format F (fixed length). All non-RRDS VSAM input/output (KSDS, RRDS, etc.) is reported as being of record format U (undefined length).

In addition to the standard, undefined length format files that employ end-of-line characters to terminate records, SELCOPY supports input/output of fixed and variable length (blocked and unblocked) record format files belonging to a hierarchical file system (e.g. FAT32, NTFS, EXT4, XFS, ZFS). Furthermore, SELCOPY supports hierarchical file system input/output of files assigned one of the following special variable length record formats. These are reported in the second LRECL column:

V2	Variable length format records that each have a leading 2-byte (4 digit) big-endian hexadecimal value defining the length of the record data that follows.
V3	Variable length format records that each have a leading 3-byte field comprising a flag byte followed by a 2-byte (4 digit) big-endian hexadecimal value defining the length of the record data that follows. Files of this format are created by FTP block mode (MODE B) transfer of z/OS RECFM VB files to ASCII based platforms.
MFV	A Micro Focus Variable length format file which has a fixed, 128-byte header followed by variable length records that each have a leading 2-byte field comprising a flag byte followed by a 1-byte (2 digit) hexadecimal value defining the length of the record data that follows.

FSIZE

For z/OS VSAM and z/VM CMS VSAM data sets, FSIZE displays the number of records in the data set when it is opened. Otherwise, FSIZE displays the number of records read from or written to a data object since that object was opened.

In z/OS and z/VM CMS environments only, if an UPDATE, DELETE and/or INSERT has been actioned on a VSAM data set, then each reference to that data set on an I/O operation within the summary block will be immediately followed by additional report lines, one each for the number of updates, deletes and inserts performed. These numbers are right adjusted in the FSIZE column and are followed by the character literal: UPD, DEL or INS.

By default, the FSIZE column starts at record position 52, has a width of 13 characters and both the column header and column values are right adjusted within the column area. If PRTSUM=3 is in effect, the column starts at record position 58.

The number of records value is displayed with comma (,) punctuation after every 3 digits starting from the units digit.

```
SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL          FSIZE   CI    DSN
 ------      ------      ----     -------  -----          -----   --    ---
    1----2        1
    3            12
    4            11 READ TESTK       100    71 U            11       DJH.TEST.KSDS
                                                              2 UPD
                                                              1 DEL
                                                              3 INS
                                    *EOF*NOT*REACHED*
```

CI

For VSAM data sets processed in a z/OS or z/VM CMS environment, this column displays the CISIZE (control interval size) value defined for the data component of the VSAM cluster.

By default, the CI column starts at record position 66, has a width of 5 characters and column values are right adjusted within the column area. The column header text is located at record position 68. If PRTSUM=3 is in effect, the column values start at record position 72 with header text starting at record position 74.

DSN

Displays the reference to the data object (fileid, SQL SELECT statement or SELCOPYi list command) to which the file name is assigned. In the case where a file name is dynamically allocated to a data object identified by a field or declared variable name, this column will display the data object to which the file name was allocated when it was last opened.

By default, the DSN column starts at record position 72 and has a width of 61 characters so restricting the summary block report record to a width of 132 characters. DSN column entries that exceed 61 characters wrap so that as many additional summary block records are written as is needed to display the full DSN entry.

Column entries are left adjusted within the column area and the column header text is located at record position 74. If PRTSUM=3 is in effect, the column values start at record position 78 with header text starting at record position 80.

```
...  BLKSIZE  LRECL          FSIZE   CI    DSN
     -------  -----          -----   --    ---
       2048   211 U          5,321       C:\nbj\ca\L05.cmx
       2048   211 U          5,321       C:\tmp\160_byte_filename_for_Gxxxx_Gxxxx_of_DDDDD_2004-12-08_
                                         sq11471_padlen7_Padding_Length-29-up-to-here-_ABCDEFGHIJKLMNO
                                         PQRSTUVWXYZ---60---abcdefghijklmnopqrstuvwxyz
```

Warning Messages

If selection time processing has completed with a non-zero return code, then warning messages are displayed in column 1 of the list output, following the diagnostics summary block.

A warning message starts with the text `***WARNING***`, and is followed by a reference to the selection identifier of the statement that triggered the warning. The format is `(SELnnnnn)`, where `nnnnn` is the right adjusted selection identifier number, padded on the left with minus/hyphen symbols (-). e.g.

```
***WARNING*** (SEL----5)        8 = RETURN CODE FROM SELCOPY
```

If the warning was triggered by a statement that is not assigned a selection identifier (i.e. an IF, AND, OR operation), then the selection identifier of the first THEN statement that follows is reported but with a single minus/hyphen (-) preceding the number. e.g.

```
***WARNING*** (SEL  -42)       52 = RETURN CODE FROM SELCOPY
```

Footer Lines

The following 2 footer lines are displayed as standard at the end of the last page of the SELCOPY List output:

```
** SELCOPY/WNT n.nn.bbb  yyyy/mm/dd  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                  ** Expiry: dd mmm yyyy **
```

Where:

- *n.nn.bbb* is the SELCOPY release and build level. e.g. 3.20.009
- *yyyy/mm/dd* is the ISO format date of this SELCOPY build level. e.g. 2017/10/17
- *dd mmm yyyy* is the date at which the active SELCOPY licence key will expire. This date is in day-of-month, abbreviated month name and 4-digit year format.

These footer lines cannot be altered but are suppressed if option NOPRINT is in effect.

Chapter 3. Data Elements and References

This chapter describes the data elements on which operations are performed within a SELCOPY program. Data elements represent the value of a constant or a variable, or the value obtained from the contents of a field.

It also describes references to data objects and data elements as terms within an expression.

Substitution Variables

Substitution variables represent values that are established and implemented during SELCOPY control statement analysis.

These types of variables have character constant values that cannot be updated during selection time (run-time) processing. They are used only as a mechanism for substituting a character constant value in place of its assigned variable name wherever that name occurs within the SELCOPY control statements that follow the assignment statement.

Substitution variables may exist as environment variables, parameter variables or equated symbols.

Environment Variables

Environment variables are a set of dynamic named values that have been established by the operating environment (shell or address space) in which the SELCOPY process is started. Environment variables (e.g. TEMP, HOME, PATH) are supported in all Microsoft Windows, Linux, Unix and OS/400 operating systems.

In z/OS TSO, SELCOPY identifies environment variable names as being MVS system symbols. Furthermore, in z/OS and CMS environments, variables established within a REXX procedure that invokes SELCOPY are also treated as environment variables.

SELCOPY supports reference to these environment variable names and substitution of their assigned values within its control statements.

SELCOPY option ENVVAR/NOENVVAR determines whether this substitution is activated or not. Unless NOENVVAR is set in the SELCNAM options file, the default action is ENVVAR (i.e. perform variable substitution.) ENVVAR and NOENVVAR options may be specified throughout the SELCOPY control statement input to activate and deactivate environment variable substitution in the control statements that follow.

If environment variable substitution is active, any text in the SELCOPY control statements, that is enclosed between two percent (%) symbols, is treated as being a reference to an environment variable name and substitution is attempted. The substitution will occur regardless of the variable name's location within a statement. e.g. within a quoted character constant.

In addition, when executing in a z/OS TSO environment, SELCOPY also interprets any text of up to 8 characters in length, prefixed by an ampersand (&) and terminated with a dot/period (.), as being an environment variable reference. e.g. the following statements are equivalent in z/OS TSO.

```
PRINT  "The System Symbol, SYSNAME, has a value of '%SYSNAME%'."
PRINT  "The System Symbol, SYSNAME, has a value of '&SYSNAME.'."
```

If SELCOPY should fail to resolve an environment variable name, then the action taken is governed by the prevailing value of option ENVFAIL. Note: Option ENVVAR will reset ENVFAIL to its default value. (i.e. SAME)

Parameter Variables

The first 9 parameters passed to the SELCOPY program as blank delimited *parm* values, are used, in sequence, as values on automatically generated EQU operations for symbols %1, %2, ... %9 respectively. The generated EQU operations are processed before all other statements in the control statement file and are displayed as the first statements of the SELCOPY list output. No EQU operation is generated for parameter zero (%0) which is always set to be the SELCOPY executable program name.

Thus, occurrences of a single decimal digit (0-9) prefixed by a percent (%) symbol anywhere within the control statements, will be substituted with the equivalent numbered input parameter value. Unlike programmer-defined equated symbols, substitution will occur regardless of the variable name's location within a statement. e.g. within a quoted character constant.

Like environment variables, the status of option ENVVAR/NOENVVAR determines whether or not substitution of parameter variables is activated. Also, if substitution is attempted for an unspecified parameter reference (e.g. %5 when only 4 parameters have been specified), then the action taken is governed by the prevailing value of option ENVFAIL.

Input parameter values may also be accessed via SELCOPY's internal copy of the parameter list which exists as a null terminated character field of minimum length 80 at the named position, PARM.

Example of parameter variable usage:

```
OPTION ENVVAR  ENVFAIL='#FAIL#'

IF  '%1'  =  '#FAIL#'
  THEN  LOG  'Error: Missing or unexpected input parameter.'
  THEN GOTO EOJ

  ELSE PRINT 'First input parameter is: "%1".'
```

Equated Symbols

Equated symbols are programmer defined names that represent character constant values.

An equated symbol name and its equated value are defined using the EQU operation which is processed during control statement analysis. All identifiers that match the name of an equated symbol and occur in control statements following the EQU operation on which the symbol is defined, are substituted with the equivalent equated value. To avoid confusion, it is recommended that all EQU operations are located at the start of control statement input.

Substitution will occur for terms within an arithmetic expression, returning a selection time error if the equated value is invalid (e.g. non-numeric). If the equated symbol has a single leading and trailing "%" (percent) character then, like environment variables, substitution may occur within a quoted character constant. Otherwise, substitution of equated symbols will not occur for text in a quoted character constant or hex character constant.

The symbol name and value defined by an EQU operation is itself subject to equated symbol substitution as defined by previously processed EQU operations. Therefore, equated values may be nested. Nesting equated symbols is a good technique to use when establishing fields within the work area buffer. e.g.

```
EQU  IREC                1    * Input  record position.
EQU  IREC_LEN          100    * Input  record maximum length.

EQU  OREC     IREC+IREC_LEN    * Output record position.       (POS 1+100)
EQU  OREC_LEN      IREC_LEN    * Output record maximum length. (100)

OPTION  WORKLEN=OREC+OREC_LEN-1      * Work area length 200.   (1+100+100-1).
```

In the above example, all operations and variable assignments may (and should) use equated symbols when referencing fields in the work area. If this is so then, should the job need to be adapted to handle input/output records of length greater than 100 (e.g. LRECL 256), then the IREC_LEN equated value need only be updated and the integrity of the job is maintained.

Substitution of equated symbols occurs before a statement is parsed by SELCOPY during control statement analysis processing. Therefore, an equated symbol name may match the name of any other identifier (SELCOPY keyword or programmer-defined name) so substituting the identifier with the value assigned to the equated symbol. Syntax analysis is then performed on the updated statement. e.g.

```
EQU   PRINT           PLOG    * PLOG (PRINT and LOG).
PRINT  "Hello World!"         * Operation keyword substituted with PLOG.
```

Warning:
Because substitution of SELCOPY identifier keywords will occur without restriction, the programmer must take care when choosing equated symbol names that are also used as keywords (or keyword synonyms) within the SELCOPY control statements.

Similarly, it is possible to define an equated symbol name comprising only numeric digits or one which includes relational or arithmetic operators (e.g. "+", "-", ">"). Doing so may adversely affect evaluation of expressions that follow and therefore should be avoided.

Work Area or Input Buffer

Every execution of SELCOPY requires a **base storage buffer** in which input data may be stored and optionally manipulated, and from which output records may be written or printed. Even if the base storage buffer is not used for these purposes, its existence is mandatory.

SELCOPY statements reference the location of a user field definition as a positional expression which evaluates to an integer value. This position value is a relative displacement which, when applied to the address of the base storage buffer (**base address**), corresponds to the address in storage of the field. Without a base storage buffer, this fundamental principle of SELCOPY operation would fail.

Work Area

The base storage buffer may be explicitly defined within a SELCOPY program using the WORKLEN option.

WORKLEN instructs SELCOPY to allocate an area of storage of static length as specified by the WORKLEN value and initialise its contents using the FILL option character (default blank). This work area becomes the base storage buffer.

By default, execution of a READ operation will copy the next record or row of data from the input buffer into which a block of data was read, to a position of the work area (default position 1). Similarly, execution of a WRITE operation will copy a field which constitutes an output record, to the next output record location within the output buffer.

Note that the work area will not be used for I/O if a declared variable of type character is specified as the target (INTO) of the READ operation or the source (FROM) of an output WRITE, PRINT, UPDATE or INSERT operation.

If the SELCOPY program has no input data object (i.e. no READ operation exists), a default work area of length 80 characters (i.e. `WORKLEN=80`) is allocated.

Input Buffer

Where option WORKLEN is not specified and an input data object exists, the base storage buffer address is the address of the record within the input buffer belonging to the last data object processed by a READ operation. The length of the base storage buffer is the length of the last input record read.

Since the address and length of the base storage buffer changes for each execution of a READ operation, the address of a user field definition changes accordingly. e.g. Position 1 will always be the address of the current input record. Therefore, unless declared variables are used, reference to the position of an input record read by a previous execution of a READ operation, is no longer possible.

Constants

A constant represents a value that does not change throughout the term of the SELCOPY program execution. Constants are specified by stating the value, where required, within the SELCOPY statements.

Character Constants

Character data is a string of one or more bytes, each representing one of the 256 characters (x'00'to x'FF') supported by the SBCS character set local to the operating system environment in which SELCOPY runs. For z/OS, z/VM CMS and OS/400 systems, character sets are based on EBCDIC encoding. For all other operating systems on which SELCOPY can run, characters sets are ASCII based.

Character constants may be used as variable and field value assignment values, as values in logical character compare operations (IF) and as source data on output operations (PRINT, WRITE). They may also be used to represent a bit mask following bitwise operators MIXED, ONES, ZEROS and on logical operations AND, OR, XOR.

A character constant may be specified in one of the following formats.

Unquoted Literals

An unquoted literal is a continuous sequence of characters that does not include an apostrophe ('), quotation mark (") or one of the blank delimiter characters. Alpha characters in unquoted literals are upper cased.

This format of character constant specification can only be used in the following:

- As assignment values for variables declared as being of one of the supported character data types.
- As assignment values for work area fields using MOD or MOVE operations.
- As test strings on IF, AND, OR operations.

Unless specified as an assignment value for an unformatted character field or a declared variable, an unquoted literal comprising only numeric digits, with or without a decimal point (.), leading unary plus (+) or leading unary minus (-), is treated as being a numeric constant.

Undesirable results may occur where an unquoted literal value is also the name of a statement identifier. e.g. A SELCOPY keyword (or keyword synonym), a declared variable name or equated symbol. To avoid ambiguity, quoted character constants should be used instead.

Examples of unquoted literals are:

Constant	Value	Length
`John`	`JOHN`	4
`a*b`	`A*B`	3
`123c`	`123C`	4

Quoted Character Constants

A quoted character constant is a contiguous sequence of characters enclosed in apostrophes (') or quotation marks ("). No upper case translation is actioned on alpha characters in quoted character constants.

Apostrophes (') or quotation marks (") may be specified as text within a quoted character constant by simply using the alternate enclosing characters. e.g. Enclose the constant in apostrophes if the constant value contains quotation marks.

Alternatively, if the enclosing character is also to be used as text within the constant, each single occurrence of that character must be entered twice. (i.e. `''` or `""`)

Examples of quoted character constants are:

Constant	Value	Length
`'a*b'`	`a*b`	3
`'No! **#---++#---**'`	`No! **#---++---#**`	18
`'She said, "John".'`	`She said, "John".`	17
`'O''Reilly''s car'`	`O'Reilly's car`	14
`''`		0

Hex Character Constants

A hex character constant is a contiguous sequence of an even number of hex digits enclosed in apostrophes (') or quotation marks (") preceded by the letter X. Each pair of hex digits represents a single character.

The length of a hex character constant is half the number of hex digits specified. Null hex character constants (`X''`) are invalid, however, any number of punctuating commas (,) may be entered between the hex digits.

A hex constant expressed in this way may also be used to represent a hex binary integer constant. Interpretation of a hex constant as either a character value or a numeric value depends on its usage within an assignment or operation statement.

If any of the following apply to the hex constant specification, it will be treated as a numeric value:

1. The hex constant is a term in an arithmetic expression. e.g.

```
1+x'20'-3    * Equivalent to:   1+32-3
```

2. The hex constant is preceded by keyword POS or is followed by keyword AT. i.e. it is an element of a field definition. e.g. The following both represent a field definition of length 16 starting at position 32 of the work area buffer.

```
POS x'20'  LEN x'10'
x'10' AT  x'20'
```

In all other instances, a hex constant will be interpreted as a character constant.

Examples of hex character constants are:

Constant	Value (ASCII)	Value (EBCDIC)	Length
`X'4D616C5D'`	`Mal]`	`(/%)`	4
`x",,,,4E,4B,5C,"`	`+.*`	`NK\`	3
`x'4C7E,406F,3B6F,407A,7E6E'`	`L~@o;o@z~n`	`<= ?,? :=>`	10

ASCII/EBCDIC Character Constants

By default, unquoted literals and quoted character constants are interpreted using the base code page encoding scheme used by the local system (EBCDIC or ASCII). e.g. the constant 'A' will be interpreted as EBCDIC X'C1' on z/OS systems, ASCII X'41' on Microsoft Windows systems.

The encoding scheme used by SELCOPY to interpret an individual character constant may be controlled by the programmer using keywords `EBC` (EBCDIC) or `ASC` (ASCII) as a suffix to the constant specification.

Examples of specific ASCII/EBCDIC character constants are:

Constant	Value	Length
`'a*b' ASC`	`X'612A62'`	3
`'a*b' EBC`	`X'815C82'`	3
`'AZaz09' ASC`	`X'415A,617A,3039'`	6
`'AZaz09' EBC`	`X'C1E9,81A9,F0F9'`	6

Numeric Character Constants

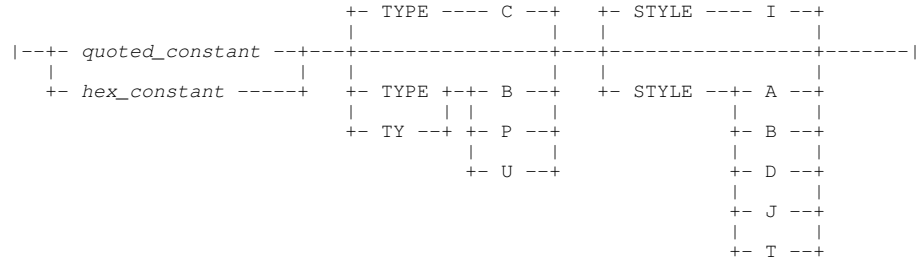
A numeric character constant is a character constant which may be interpreted by SELCOPY as a numeric constant. A numeric character constant may be used on operations and variable assignments in place of a numeric constant. If the numeric constant contains blank characters or "," (comma) punctuation, it must be specified as a quoted character constant.

The format of a numeric character constant and its interpretation by SELCOPY is discussed in detail under Numeric Character Data.

Date Character Constants

Applicable only to CVDATE operation source values, a date constant represents a date value in a format specified by a TYPE and STYLE specification.

See Date Data Types for detailed information on the types and styles of date supported by SELCOPY.



Parameters:

quoted_constant | hex_constant
A quoted character constant or hex character constant that specifies the source date value. The interpretation of this date value depends on the TYPE and STYLE option attributed to it.

A character (TYPE=C) source value may contain numeric and non-numeric digits. The format of a TYPE=C source date value and its interpretation by SELCOPY is discussed in detail under character date data type.

The format used to specify the date value may be selected so that data best suits the TYPE attribute. i.e. *quoted_constant* is preferable when TYPE=C is used, *hex_constant* for TYPE=B, P and U.

STYLE
STYLE identifies one of the following commonly used date styles (date format).

A	American standard date format. (mmddyyyy)
B	British (and European) standard date format. (ddmmyyyy)
D	Number of days since 1900/01/01.
I	International (ISO) standard date format. (yyyymmdd)
J	Julian date format. (yyyyddd)
T	8-byte binary time-of-day (TOD) clock as obtained via the z/Architecture STCK instruction. Any TYPE specification is ignored for STYLE=T.

TYPE
TY
TYPE specifies the source data type of the date value.

B	The decimal numerical digits that constitute the date value expressed as a signed binary integer value. e.g. 2015/08/21 (STYLE=I) is X'01337A25' (decimal 20150821).
C	Character date. A character date source value may contain numeric and non-numeric digits. The format of a TYPE=C source date value and its interpretation by SELCOPY is discussed in detail under character date data type.
P	The decimal numerical digits that constitute the date value expressed as a signed packed decimal integer value. e.g. 2015/08/21 (STYLE=I) is X'020150821C'.
U	The decimal numerical digits that constitute the date value expressed as an unsigned packed decimal integer value. e.g. 2015/08/21 (STYLE=I) is X'20150821'.

Examples:

Examples of date character constants are:

Constant	Date Value (FORMAT='yyyy/mm/dd')
'12/25/2012' TYPE=C STYLE=A	2012/12/25
'2003/243' TYPE=C STYLE=J	2003/08/31
'34555' TYPE=C STYLE=D	1994/08/10
X'1E8482' TYPE=B STYLE=J	2000/01/02
X'2013,142C' TYPE=P STYLE=J	2013/05/22
X'1806,2005' TYPE=U STYLE=B	2005/06/18
X'92EE35A6,DE000000' STYLE=T	1981/11/29

Numeric Constants

Numeric constants are values that represent a quantity, length or location within SELCOPY syntax.

Numeric constants are the simplest form of arithmetic expressions used within SELCOPY syntax. Multiple numeric constants may, of course, be used as terms within more complex arithmetic expressions.

A numeric constant may be specified in one of the following formats.

Decimal Integer Constants

A decimal integer constant is comprised of one or more contiguous decimal digits (0-9) optionally prefixed by a unary plus (+) or minus (-) operator. Decimal integer constants must not contain comma punctuation or a decimal point. If no leading unary plus or minus symbol is specified, unary plus is assumed by SELCOPY.

A decimal integer constant has a precision (p), where p is the total number of digits in the constant.

A decimal integer constant is treated as an unquoted literal character constant when specified as the value assigned to a field or declared variable of data type character with no numeric interpretation.

Examples of decimal integer constants are:

Constant	Precision
5	1
+132	3
-23	2
32768	5
0006	4
0	1

Zoned Decimal Integer Constants

A zoned decimal integer constant is comprised of one or more decimal digits (0-9) and ends with one of the upper case alpha characters "C", "D" or "F". Zoned decimal integer constants must not contain comma punctuation.

If the base code page encoding scheme used by the local system is ASCII, a zoned decimal constant referenced by a SELCOPY operation will be converted to EBCDIC before being processed.

Each digit of the zoned decimal value is represented by one byte of the EBCDIC representation of the constant value. The right-most 4 bits of each EBCDIC byte denote a number which is the zoned decimal digit value represented as a hex code (x'0'-x'9').

For all the EBCDIC bytes except the last, the left-most 4 bits of the byte denote the zone. The zone bits contain a fixed hex code (x'F') so that the zone and number bits together constitute a printable EBCDIC character value corresponding to the decimal digit. i.e. X'F0' to x'F9' is EBCDIC characters 0 to 9.

The left-most 4 bits of the last EBCDIC byte denote the value's sign. Hex codes x'C' and x'F' indicate a positive zoned decimal value (+) and code x'D' indicates a negative value (-). Note that, although zoned decimal values can have a sign hex code in the range x'A' to x'F', SELCOPY only supports zoned decimal values with the industry preferred sign codes: x'C', x'D' or x'F'.

A zoned decimal integer constant has a precision (p), where p is the total number of digits in the constant.

A zoned decimal integer constant is treated as an unquoted literal character constant when specified as the value assigned to a field or declared variable of data type character.

Examples of zoned decimal integer constants are:

Constant	EBCDIC Representation	Value	Precision
B	x'C2'	+2	1
72H	x'F7F2C8'	+728	3
121	x'F1F2F1'	+121	3
0000J	x'F0F0F0F0D1'	-1	5
03D	x'F0F3C4'	+34	3

Hex Binary Integer Constants

A hex binary integer constant is a '0x' or '0X' followed by a minimum of 1 and a maximum of 8 hex digits (0-F). Regardless of the processor architecture on which SELCOPY is running, a hex binary integer constant is interpreted as being a signed, 4-byte, big-endian, binary value of precision 31.

When less than 8 hex digits are used, the stored value will be padded to 4-bytes using the most significant (sign) bit of the specified hex binary integer. The exception to this is when a hex binary integer is expressed as only 1 or 2 hex digits (i.e. 1-byte), in which case the value is treated as being unsigned and the stored value is padded with zeroes.

See also Hex Character Constants for alternate specification of a hex binary integer constant value.

Examples of hex binary integer constants are:

Constant	Same as	Decimal Value
0x22	0x00000022	+34
0x1AD	0x000001AD	+429
0x0A3F	0x00000A3F	+2623
0xA3F	0xFFFFFA3F	-1473
0xF	0x0000000F	+15
0xFF	0x000000FF	+255
0xFFF	0xFFFFFFFF	-1

Decimal Fixed Point Constants

A decimal fixed point constant is comprised of one or more decimal digits (0-9) with a decimal point, optionally prefixed by a unary plus (+) or minus (-) operator. Decimal fixed point constants must not contain comma punctuation. If no leading unary plus or minus symbol is specified, unary plus is assumed by SELCOPY.

A decimal fixed point constant has a precision (p,s), where p is the total number of digits in the constant and s is the scale (number of fraction digits).

A decimal fixed point constant is treated as an unquoted literal character constant when specified as the value assigned to a field or declared variable of data type character.

If a decimal fixed point constant is used in SELCOPY syntax where only an integer value is appropriate, then the fraction digits are ignored. e.g. when specifying a storage offset or length value.

Examples of decimal fixed point constants are:

Constant	Precision
5.4	(2,1)
+2.592	(4,3)
-239.88	(5,2)
32768.43730	(10,5)
.0006	(4,4)
0.01	(3,2)

Zoned Decimal Fixed Point Constants

A zoned decimal fixed point constant is comprised of one or more decimal digits (0-9) with an optional decimal point (.) and ends with one of the upper case alpha characters "C", "D" or "F". Zoned decimal fixed point constants must not contain comma punctuation.

See zoned decimal integer constants for interpretation of zoned decimal digits.

A zoned decimal fixed point constant has a precision (p,s), where p is the total number of digits in the constant and s is the scale (number of fraction digits).

A zoned decimal fixed point constant is treated as an unquoted literal character constant when specified as the value assigned to a field or declared variable of data type character.

Examples of zoned decimal fixed point constants are:

Constant	EBCDIC Representation	Value	Precision
0.1A	x'F04BF1C1'	+0.11	(3,2)
2.G	x'F24BC7'	+2.7	(2,1)
12.1	x'F1F24BF1'	+12.1	(3,1)
32.1J	x'F3F24BF1D1'	-32.11	(4,2)
666.666N	x'F6F6F64BF6F6F6D5'	-666.6665	(7,4)

Variables

A variable represents a value that may change during selection time processing of the SELCOPY program execution. Contrast this with substitution variables that are implemented during control statement analysis.

The values assigned to SELCOPY internal variables and @Variables may be updated automatically by SELCOPY when operations are executed that relate directly to the variable. For all types of variable, the value may be updated by the programmer via a variable assignment statement.

Variable values are referenced by coding the variable name, where required, within the SELCOPY statements.

The three types of variables supported by SELCOPY are: Declared, Internal and @Variables.

Declared Variables

Declared variables are programmer defined names that represent a variable value of specific data type.

Before it can be referenced in an operation or variable assignment, a declared variable must first be declared via one of the following methods:

- Using the DECLARE operation. Because DECLARE is processed during control statement analysis, it is recommended that all DECLARE operations are located at the start of control statement input.
- Automatically by SELCOPY as a result of a READ operation on a SELCOPYi list or on a database (or equivalent) object accessed via ODBC. A variable of data type character is declared for each column of input text with a name equivalent to the input column name.

A declared variable references a value which, as for field definition values, is determined from its address (position) in storage, its length (precision) and its data format (data type). Any reference to the variable name in SELCOPY control statements is a reference to that variable's current value.

Source data types supported for declared variables are documented under Data Types.

Storage Remap

By default, a declared variable references a value defined in a source field within a dynamic area of storage.

Alternatively, the address of the variable source field may exist within storage that has already been allocated (e.g. the work area or another declared variable). This provides a means by which input data fields can be mapped to variable names, and is the method used by SELCOPY when automatically declaring column name variables for ODBC database and SELCOPYi list input. For variables declared via a DECLARE operation, this is achieved using the **POS** parameter keyword. e.g.

```
DECLARE   IREC         CHA(256)
DECLARE   ID_NUM       BIN(2)     POS IREC
DECLARE   ID_LASTNAME  CHA(20)    POS IREC+002

READ INDD    INTO IREC
```

Initial Value

A variable declared via the DECLARE operation, is assigned an initial value as specified by the **INI** parameter. If no INI parameter is specified, variables of character data type are initialised with blank characters and variables of numeric data type are initialised as zero (0). This occurs regardless of whether the variable is used to map a field in pre-allocated storage. e.g.

```
DECLARE   E159      CHA(10)    INI='ERR159'
DECLARE   MAX_AGE   BIN        INI=65        POS IREC+53
```

&*varname* Source Field Position

The prevailing value assigned to any declared variable is stored in that variable's source field. The raw data in this field exists in a format which corresponds to the data type assigned to the variable.

Within SELCOPY control statements, a declared variable name may be prefixed by special character ampersand (**&**) in order to reference the position of that variable's source field.

Any reference to a declared variable of a character data type which does not have an associated numeric attribute (FORMAT), is a reference to the first position of text data in that variable's source field. Therefore, for non-numeric character variables, use of an & prefix is supported but unnecessary.

However, a reference to a declared variable of numeric data type (including a variable of character data type assigned a numeric FORMAT) is interpreted as being a numeric value. This value is determined by the variable's source field data type and the raw data stored within its limits. Therefore, unlike character type variables, in order to reference the source field position of a numeric (or character numeric) declared variable as opposed to its value, the variable name must be prefixed by &. e.g.

```
DECLARE ABC  BIN(4)  INI '16,909,060'         * Which is x'0102,0304'.

PRINT   ABC                * Will print the value held in ABC, formatted using the default 14 byte
                           * FMT='SS,SSS,SSS,SS9' giving the 14 byte string:  ' +16,909,060'

PRINT  &ABC  FMT='xxxx,'   * Will print the source field for ABC, length 4, using the hex format
                           * provided, giving:  '0102,0304'
```

Note that offsets within a declared variable source field may be referenced using an arithmetic expression where the position of the source field is the first term. e.g. &ABC+1

Internal Variables

Internal variables are variables of pre-determined name that reference values maintained in specific SELCOPY internal fields. The field values relate to processing information and are updated by SELCOPY over the course of a program's execution.

All of SELCOPY's internal variables represent integer values and most may be referenced by name as terms within an arithmetic expression. These variable fields are of binary integer data type and have internal storage locations that may be referenced via UX prefixed position names.

Although maintained by SELCOPY, some internal variable values may be updated by the program via a variable assignment statement (e.g. LRECL=80). Beware that doing so will have an effect on SELCOPY's default processing and that these values are subject to change by SELCOPY. e.g. The LRECL value will be updated following the next execution of a READ operation.

The complete list of internal variables including the variable name, the related internal field definition and its description are detailed in the following table. Unless otherwise stated, these internal variables may be updated via a variable assignment statement and may be specified as terms within an arithmetic expression.

Variable Name	Field Name	Description
DIFF	UXADIFF	Following a character compare operation that is not a range test, this variable denotes the first position within the first character field at which a difference was found. If no difference was found, this value is 0. Internal field UXADIFF actually contains the storage address of the difference. This address is equivalent to the base address (address of the work area or current record within the input buffer) plus the value of DIFF minus 1.
INCOUNT IN	UXINCNT (Value applies only to the prime input.)	Maintained as a number of values, one for each input data object, INCOUNT references the count of input records obtained as a result of a READ operation. INCOUNT cannot be updated via direct assignment nor included as a term in an arithmetic expression. However, the INCOUNT value of a particular data object may be tested using an arithmetic comparison operation. e.g. `IF INCOUNT INDD > 20  * Test for more 20 input records from file name INDD.`
LINE	UXLINE	A value between 1 and the defined page depth, this variable represents the line number of the next line to be written to the SELCOPY list output. Assigning a value that is less than the current value will throw a new page. The assigned LINE value will not represent the next output line number of the new page, but will immediately be updated by SELCOPY to reflect the output line number following the page headers. LINE cannot be included as a term in an arithmetic expression.

LRECL L	UXLRECL	The length of the last input record obtained as a result of a READ operation. This value is used as the default length of a field referenced by position only. e.g. `WRITE OUTDD FROM 101 * Field starting at position 101, length LRECL.`
REASCD	UXREASCD	Applicable to z/VM CMS only, the reason code returned by a CMS function, in addition to the CMS return code (RETSYS).
RETCD RETCODE RC	UXRETCD	SELCOPY's return code. This value contains the highest return code as set by the SELCOPY program or by a RETCODE variable assignment statement. A non-zero SELCOPY return code will trigger a SELCOPY warning message.
RETSYS RETCMS RETXV CMSRETCD	UXRETSYS	The return code set by the last system command/operation executed for a SELCOPY CP, SYSTEM, UTIME or XV operation.

@Variables

Originally introduced as pointers to positions in the work area or input buffer storage, @variable programmer defined names (also known as @pointer variables) represent integer values that are stored internally as signed, binary integer fields.

By default, an @variable value is assigned automatically by SELCOPY following a character compare range test. Unless parameter keyword PTR is specified on the IF operation, the name of this @variable is @ (i.e. symbol "@" without a name suffix.)

```
IF  POS 11, 256  =  'Hello'          * Range test.
  THEN PRINT  FROM  @,  @+11         * @ --> 'Hello World.'
```

Following a successful range test, the @variable is assigned a value equal to the position of the matched text within the range. This is expressed as 1 plus the offset of the matched text from the base address (i.e. address of the work area or input record buffer). In the above example, if the character value "Hello" is located at position 89 of the work area, then the value of @ following the range test will be 89.

If the range test is not successful in identifying a match, a value of zero (0) is assigned and the @variable is also flagged as being NULL. Note that, since an @variable may be assigned a zero value, the NULL flag is necessary to indicate that the @variable is unset (IF @abc = NULL).

@Variables may also be explicitly assigned to signed integer values via a variable assignment statement, and may be specified as terms within an arithmetic expression.

```
@offset = -5
MOD  POS INREC+LRECL+@offset-1  =  '-x#x-'    *  Last 5 chars of input record.
```

The storage address referenced by POS @ (the default @variable) is maintained by internal field UXATPTR. Any update to this internal field will also update the value assigned to @.

Field Definitions

Field definitions may be used in individual SELCOPY operations or variable assignments to reference data in source and/or target areas of storage.

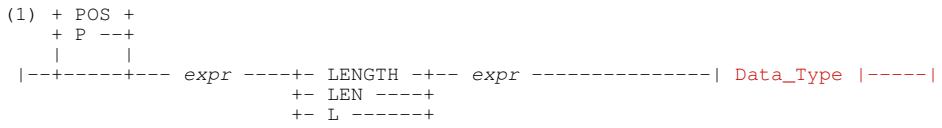
Like declared variables, a field has location (position), length and data type attributes. Though, unlike declared variables, a complete field definition must be specified wherever it is used and may be variable, i.e. referencing a variable position in storage or have a variable length.

Fields may be expressed in SELCOPY command syntax using one of the following field definition types. Note, however, that not all SELCOPY operations support the full range of field definition types. Where this is true, the supported field definition types are identified in the description of the operation.

The position of a field, referenced by the expression *expr*, resolves to be an address in storage when processed by SELCOPY. The value of *expr* is a positive or negative integer value that corresponds to an offset from a base storage address. This base address is the address of the work area or, if no work area has been specified, the address of the current record within the input buffer belonging to the prime input file. The offset applied to the base address is equal to the value of *expr*-1. Therefore, POS 1 is offset zero (0) from the base address (i.e. position 1 of the work area or prime input record.)

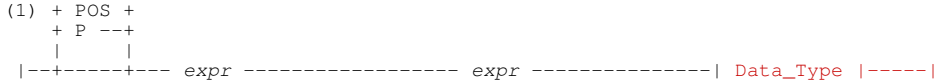
Field Type 1:

Field definition specified as a storage position and length (***field_pLENn***).



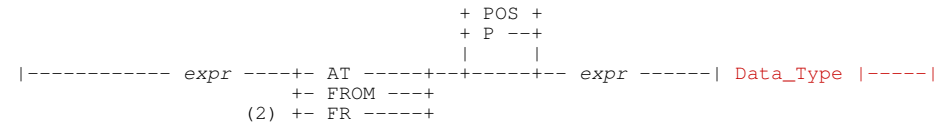
Field Type 2:

Field definition specified as start and end positions in storage (***field_p1p2***).



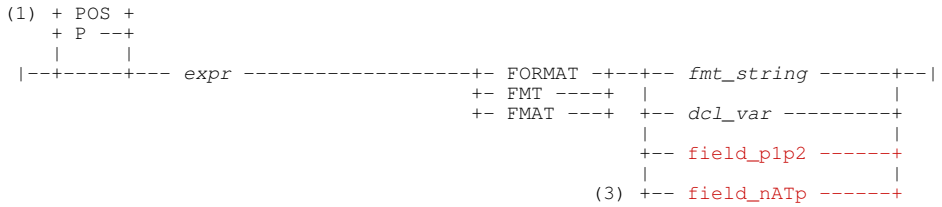
Field Type 3:

Field definition specified as a length, starting at a storage position (***field_nATp***).

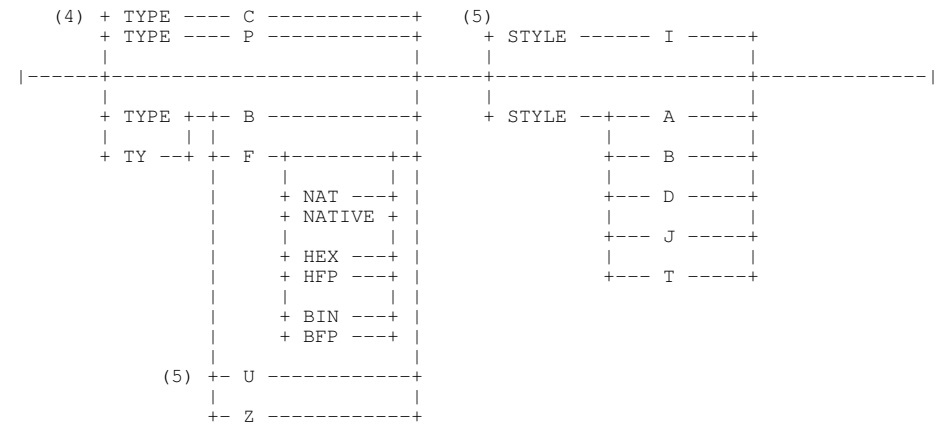


Field Type 4:

Field definition specified as a storage position with a length equal to the length of a format string (***field_pFMT***).



Data_Type:



Syntax Notes:

(1) Specification of the POS keyword may be mandatory. See parameter POS for details.

(2) Restrictions exist for use of synonym FROM and FR. See parameter AT for details.

(3) *Data_Type* will be ignored if specified on *field_p1p2* or *field_nATp* field definitions which represent a FORMAT string.

(4) The default data type depends on the operation. See parameter TYPE for details.

(5) STYLE and TYPE=U are applicable only to date fields used as the source or target of a CVDATE operation.

Parameters:

AT
FROM
FR

Identifies the field definition as Type 3.

The mandatory *expr* expression that precedes the AT or FROM keyword defines the length of the field and the mandatory *expr* that follows defines the position in storage of the first byte of the field.

Synonyms FROM and FR are invalid for the following uses of Type 3 fields:

1. As a target of an operation, following TO or INTO keyword parameters.
2. On operations where FROM is used to identify one or more source fields. e.g. MOD, MOVE, WRITE and PRINT.

e.g.

```
PRINT  FROM  4 FROM 1                 * Prints 2 source fields at POS 4 and 1.

MOVE         4 FROM 1  TO  4 AT   21  * OK.
MOVE   FROM  4 AT   1  TO  4 AT   21  * OK.
MOVE   FROM  4 FROM 1  TO  4 AT   21  * Invalid - ERROR 045.
MOVE         4 FROM 1  TO  4 FROM 21  * Invalid - ERROR 045. (FROM in target)
```

expr

An arithmetic expression which evaluates to a positive, non-zero, integer value, *expr* may identify a length value or a position within the work area/input record buffer.

expr may identify a field position outside the work area or input record buffer if it includes any of the following:

◊ A declared variable of a character data type with no numeric interpretation (i.e. no FORMAT).
◊ The source field of a declared variable of any data type, specified as &*varname*.
◊ A named SELCOPY internal field position. e.g. RPL
◊ An @ variable whose value is based on a named SELCOPY internal field position. e.g. @abc = DATE-2
◊ An @ variable whose value has been assigned indirectly via an update to the UXATPTR field.

For a Type 2 field, the start and end positions, represented by 2 **expr** values, must define an ascending sequence of storage locations. If not, then either a control statement error (ERROR 074) or, if the relative positions are variable, return code 8 is set.

FORMAT *fmt_string* | *dcl_var* | *field_p1p2* | *field_nATp*
FMT

Identifies the field definition as Type 4, a character field containing a numeric value. Alternatively, it specifies the format of printable hex on a CVCH operation.

The FORMAT string value is a template which determines the character representation of the numeric or hexadecimal digits. Type 4 field definitions are used as the target field specification of a data conversion or modification operation (CVxx, MOD).

The FORMAT string value is specified as a quoted character constant (*fmt_string*). Alternatively, if specified on the target of a CVxC or CVCH operation, the FORMAT string value may be specified as a declared variable of character data type (*dcl_var*) or a Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character (TYPE=C) field definition. Any specification of *data_type* on a FORMAT string specified as *field_p1p2* or *field_nATp* will be ignored.

See format specification parameters for information on *fmt_string* syntax.

LENGTH
LEN
L

Identifies the field definition as Type 1. Note that not all operations (e.g. ADD, SUB, MULT, DIV) support use of Type 1 fields.

The mandatory **expr** expression that follows defines the length of the field. If no AT, FORMAT, LENGTH or *expr* is specified following the first *expr* of the field definition, then the length of the field is derived as follows:

◊ For a field that represents either the target or source of a MOD or MOVE operation, the default field length is equivalent to the length of the field, constant or declared variable specified as the source or target of the operation respectively. e.g.

```
MOD   POS 101  =  POS  1  LENGTH 20  * Target field is:  POS 101  LENGTH 20
MOD   POS @A+1 =  'OK'               * Target field is:  POS @A+1 LENGTH  2
MOVE  POS   1  TO   CHA1             * Source field has CHA1 variable length.
```

If length cannot be determined from either of the target or source fields, then a control statement error (ERROR 069) is returned.

◊ For READ and UPDATE operations, specification of length on the input (INTO) target field or update (FROM) source field is invalid. The length of the field is the length of the record read from the nominated input data object.

◊ For WRITE, INSERT, PRINT and PUNCH operations, the length of the output (FROM) field defaults to be the current value of the internal variable LRECL. However, if the output data object is of a fixed length, the length of the output field defaults to be the length assigned to the data object.

◊ For CVCH and CVHC operations, specification of length on the target (TO) field is ignored. The length of the field is either double (CVCH) or half (CVHC) the length of the source field.

◊ For all other operations where a Type 2 fields are supported, a length specification is mandatory and so a control statement error (ERROR 069) is returned.

POS
P

Identifies the field definition as either Type 1, Type 2 or Type 4.

The mandatory *expr* expression that follows defines the location of the first byte of the field. If *expr* is followed by another *expr*, then a Type 2 field definition is assumed where the second *expr* defines the position in storage of the last byte of the field.

Specification of POS is mandatory in the following cases:

1. The field is the target of a MOD operation for which the optional MOD operation keyword has not been specified. e.g.

```
POS 51  =  'ABC'              * Equivalent to:  MOD  POS 51 LENGTH 3  = 'ABC'
```

2. The field is one of the terms in an IF, OR, AND compare operation. e.g.

```
IF POS 1 = POS 22, 27         * Equivalent to:  IF POS 1 LENGTH 6 = POS 22, 27
```

3. The field position, *expr* references a numeric declared variable as the starting position within the work area from which the remaining terms of the expression define an offset. Alternatively, if *expr* references a starting position in the work area as a numeric integer constant from which a numeric declared variable defines an offset. e.g.

```
DECLARE  B1 BIN  INI=  5      * Declared binary variable, B1 = 5.
@OFF             =  8      * @variable, @OFF = 8.

MOD   POS   B1+@OFF = 'OK'    * Equivalent to:   MOD  POS 13 LENGTH 2  = 'OK'
MOD   POS 5+B1      = 'X'     * Equivalent to:   MOD  POS 10 LENGTH 1  = 'X'
IF    POS   B1      = '123'   * Equivalent to:   IF   POS  5 LENGTH 3  = '123'
```

STYLE

Applicable only on field definitions specified as the source or target of a CVDATE operation, STYLE identifies the style of date. A date field may be of binary, character, signed or unsigned packed decimal data type. See TYPE=B, C, U and P.

A	American standard date format. (mmddyyyy)
B	British (and European) standard date format. (ddmmyyyy)
D	Number of days since 1900/01/01.
I	International (ISO) standard date format. (yyyymmdd)
J	Julian date format. (yyyyddd)
T	8-byte binary time-of-day (TOD) clock as obtained via the z/Architecture STCK instruction. Any TYPE specification is ignored for STYLE=T.

See Date Data Types for detailed information on the styles of date supported by SELCOPY.

TYPE
TY

Applicable only on Type 1, Type 2 and Type 3 field definitions, TYPE specifies the data type of the field. If specified on a Type 4 field definition, the TYPE parameter is ignored.

B	Hexadecimal data representing a signed binary integer value.
C	Character data. If a field defined as TYPE=C is used as the destination of a binary, packed decimal or floating point value conversion, then TYPE=C is equivalent to TYPE=Z.
F	Hexadecimal data representing a floating point value. TYPE=F sub-parameters define the format of the floating point data with the default defined by environment option DEFAULTFP. **BIN** | **BFP** IEEE-754 Base 2 Binary. **HEX** | **HFP** IBM Base 16 Hexadecimal. **NATIVE** | **NAT** Floating point format native to the local machine architecture.
P	Hexadecimal data representing a signed packed decimal integer value.
U	Applicable only to date fields identified in a CVDATE operation, TYPE=U identifies hexadecimal data representing an unsigned packed decimal date value.
Z	Character data representing a zoned decimal integer value.

Default field data type is determined as follows:

1. For data type conversion operations (CVxx/CVDATE), the data type of the source and target fields are implied by the operation keyword. e.g. CVBC treats the source field as TYPE=B and the target field as TYPE=C.
2. For arithmetic operations (ADD, SUB, MULT, DIV), the data type is the same as the first operand in the operation for which a data type is defined. This may be a field with a TYPE specification or a declared variable. If none of the operands have an associated data type, then **TYPE=P** is default. e.g.

```
ADD  4 AT 1   TO  4 AT 21 TYPE=B   INTO  4 AT 101 TYPE=Z   * Default TYPE=B.
```

3. For the GENERATE operation, **TYPE=P** is default.
4. For all other operations, **TYPE=C** is default.

See Data Types for detailed information on the data types supported by SELCOPY.

Notes:

1. *Return Code 8:*
   Any SELCOPY operation performed on a field which is located at a position within the work area or input record buffer, will give a return code of 8 if the field occupies a position that falls outside the allocated work area buffer.
2. *Equated symbols:*
   Equated symbols may be used to represent any of these field definition types or any syntax element which constitutes part of the field definition (e.g. *expr*). However, the equated symbol is still bounded by any restrictions imposed on the field definition type it represents. e.g. The following gives an error because an implied MOD operation is invalid for Type 3 field definitions.

```
equ  fldty1  pos 1  len 4  type=p  * Type 1 field definition.
equ  fldty3    4 at 1      type=p  * Type 3 field definition.
fldty1  =  23                      * OK.
fldty3  =  23                      * Returns ERROR 042.
```

Internal Field Definitions

SELCOPY supports a number of internal field definitions that may be used in individual SELCOPY operations or variable assignments.

These fields have named positions and are maintained by SELCOPY and may alter over the course of a SELCOPY program. The positional name, length and data type of each internal field follows:

ARG

POS ARG is the position of a null terminated character string field of minimum length 80, which contains the complete program parameter (argument) string as received by the SELCOPY executable. This includes any parameter strings, command line control statements, control statement input file and/or report file output file specification on the SELCOPY invocation.

If the length of the argument string is less than 80, the string is padded with blanks up to 80 characters and the null terminator (x'00') immediately follows the 80th character. If the length is longer than 80, the null terminator occurs immediately following the last character of the argument string.

In the following invocation of SELCOPY...

```
SELCOPY 'Parm 1' 'Parameter2'  -ctl /home/xuser/ssparm01  -lst /tmp/ssparm01.lst
```

...the contents of the field at POS ARG are:

```
'Parm 1' 'Parameter2'  -ctl /home/xuser/ssparm01  -lst /tmp/ssparm01.lst
```

See also internal field PARM for input parameter strings only.

CBLNAME

Only applicable to SELCOPY in z/OS and z/VM CMS environments, POS CBLNAME is the position in storage of the loaded CBLNAME options module.

The CBLNAME options module is a structure containing a number of fields of different length and data type. The CBLNAME structure includes one or more variable length extensions, one for each licensed product element (SELCOPY, SELCOPYi and CBLVCAT). The format of option flags and fields in this structure, may be derived from the CBLNAME assembly listing which generates a CBLNAME DSECT from the CBLNAME Assembler macro.

The CBLNAME structure fields may be used for reference purposes only. Updates to flags bits or field data within the structure will have no affect on the current (or any subsequent) execution of SELCOPY.

DATE

POS DATE is the position of the 8 character, ISO date format field within SELCOPY's DATE control block structure (yy/mm/dd).

The position, expressed as an offset from position DATE, length and data type of fields in the DATE control block structure are as follow:

Position	Length	Type	Example	Description
DATE-36	4	Binary	x'559B,9F6B'	Number of seconds since 1970/01/01 00:00:00
DATE-32	4	Binary	x'0000,03B9'	Number of milli-seconds (1/1000 second) after the second.
DATE-28	4	Packed Decimal	x'2015,188F'	Julian Date - `X'YYYY,DDDF'` (where *"YYYY"* and *"DDD"* are the "year" and "day of year" number respectively)
DATE-24	4	Unsigned Packed Decimal	x'0104,4110'	Time of Day - `X'0hhm,msst'` (where *"hh"*, *"mm"*, *"ss"* and *"t"* are "hours", "minutes", "seconds" and "tenths of second" respectively)
DATE-20	8	Character	'07/07/15'	USA Date - `'MM/DD/YY'` (where *"MM"*, *"DD"* and *"YY"* are "month of year", "day of month" and "year of century" number respectively)
DATE-12	1	Character	' '	Filler - blank character.
DATE-11	8	Character	'07/07/15'	European Date - `'DD/MM/YY'` (where *"DD"*, *"MM"*, and *"YY"* are "day of month", "month of year" and "year of century" number respectively)
DATE-03	1	Character	' '	Filler - blank character.
DATE-02	2	Character	'20'	Century - `'CC'` (where *"CC"* is "century" number)
DATE+00	8	Character	'15/07/07'	ISO Date - `'YY/MM/DD'` (where *"YY"*, *"MM"* and *"DD"* are "year of century", "month of year", "day of month" number respectively)
DATE+08	1	Character	' '	Filler - blank character.
DATE+09	10	Character	'10:44:11.9'	Time of Day - `'hh:mm:ss.t'` (where *"hh"*, *"mm"*, *"ss"* and *"t"* are "hours", "minutes", "seconds" and "tenths of second" respectively)
DATE+19	1	Character	' '	Filler - blank character.
DATE+20	9	Character	'Tuesday '	Day of Week - `'Weekday'` (where *"Weekday"* is the blank padded day name in mixed case)
DATE+29	1	Character	' '	Filler - blank character.
DATE+30	4	Character	' 7th'	Day of Month - `'nnst'`, `'nnnd'`, `'nnrd'` or `'nnth'` (where *"nn"* is the "day of month" number)
DATE+34	1	Character	' '	Filler - blank character.
DATE+35	9	Character	'July '	Month of Year - `'Month'` (where *"Month"* is the blank padded month name in mixed case)
DATE+44	1	Character	' '	Filler - blank character.
DATE+45	8	Character	'2015/188'	Julian Date - `'YYYY/DDD'` (where *"YYYY"* and *"DDD"* are the "year" and "day of year" number respectively)
DATE+53	1	Character	' '	Filler - blank character.
DATE+54	5	Character	'Wk:27'	Week of Year - `'Wk:ww'` (where *"ww"* is 00-52, the "week of year" number) Sunday is day 1 of the week. If 1st January falls on Thursday, Friday or Saturday (day 5, 6 or 7), the first week of the year is week 0 (Wk:00), otherwise it is week 1 (Wk:1).
DATE+59	1	Character	' '	Filler - blank character.
DATE+60	4	Binary	x'0000,A4CF'	Number of days since 1st Jan 1900.
DATE+64	4	Binary	x'0000,96FB'	Number of seconds since midnight (00:00:00).
DATE+68	4	Binary	x'000E,8AA8'	Number of micro-seconds (1/1000000 second) after the second.
DATE+72	4	Binary	x'0000,03E8'	Elapsed milli-seconds (1/1000 second) since the start of the program.

DSN

POS DSN is the position of a fixed length 255 character field containing the name of the last input or output data object processed by SELCOPY during selection time processing.

For input and output files (data sets), this is a full fileid (DSN) of the file processed. i.e. For hierarchical file systems, it is the complete fileid reference including disk letter (where applicable) and directory path. For native z/OS data sets, it is the DSN with TSO or Security Manager (ACF) userid prefix and parenthesised member name if applicable. For native z/VM CMS files, it is

the filename, filetype and filemode with single intervening blank characters.

For SELCOPY I/O operations that perform dynamic allocation, the DSN field contains the fileid as specified on the DSN parameter of the READ or WRITE operation. For SELCOPYi list or ODBC database input, the DSN field contains the SELCOPYi list command, SQL query (SELECT) statement or database table name as specified on the READ operation LIST, SQL or TABLE parameters respectively.

Dynamic Allocation
The DSN field value for a particular data object is established when the object is opened. Therefore, when the argument on a DSN, LIST, SQL or TABLE parameter is a field definition, then an update to this field and subsequent re-open of the associated file name (**fname**) reference, will also update the contents of the DSN field.

DIR/DIRDATA Input
The DSN field does not reflect the full fileid of data objects read with the DIR or DIRDATA parameter. For DIR or DIRDATA input of z/OS PDS or PDSE members, the DSN field contains the DSN of the PDS/PDSE library only. For DIR or DIRDATA input of z/VM CMS or hierarchical file system files, the DSN field contains the generic fileid with wildcards, as specified on READ statement.

The PDS/PDSE member name, CMS file name or hierarchical file system fileid being processed at any time during selection processing, may be established via the directory input record. For DIRDATA input, the data object's directory record (as opposed to one of its data records) may be identified using an IF DIR test executed following a READ DIRDATA operation on the associated fname.

Concatenation
The CAT sub-operation may be specified following a READ operation to concatenate input data objects. Following concatenated data object input, the contents of the DSN field are updated to reflect the fileid (DSN), SELCOPYi list command, ODBC SQL query statement or database table name of the particular data object from which the record was read.

FHDR

POS FHDR refers to the position of one of the following:

1. A field containing the table column headers of the last input database result table read using ODBC or the last SELCOPYi list input object. The specified (or default) separator character is used to separate each column name in this FHDR field. The field is of variable length equal to the LRECL of input data.

   Immediately following the generated column headers field is a field of equal length which contains the underlining for the headers field. This comprises minus signs, except for the separator characters which match those in the header field.

2. A fixed length 128 character field containing the file header record of the last Micro Focus variable length format file (RECFM=MFV) processed by SELCOPY.

   Processing of Micro Focus variable length format files will automatically bypass this header record on input and automatically generate a header record on output.

FNAME

POS FNAME is the position of a fixed length 8 character field containing the file name (fname) assigned to the last data object processed by SELCOPY.

Concatenation
The CAT sub-operation may be specified following a READ operation to concatenate input data objects. Following concatenated data object input, the contents of the FNAME field are updated to reflect the file name associated with the particular data object from which the record was read.

FSIZE

POS FSIZE is the position of a 4-byte binary field containing the size of the last file processed for input or output.

For files belonging to a hierarchical file system, the file size is equal to the number of **bytes** in the file at the time it was opened. For all other types of data object, the file size value is zero (0).

HEAD

POS HEAD is the position of a fixed length 156 character field containing the text of the first header line displayed at the top of each page in the SELCOPY list output. HEAD+160 is the position of the fixed length 156 character field containing the hyphen (-) underline characters of the second header line. Each hyphen (-) is at an offset of 160 characters from the character under which it appears in the report output.

The default HEAD field title text identifies the SELCOPY product version and licensed entity details and is followed by the execution timestamp and page number. However, the header text and corresponding underline output may be initialised via the HEAD environment option and later changed during selection time processing via updates to the HEAD field.

PARM

POS PARM is the field position of a null terminated character string of minimum length 80, which contains only the parameter string as received by the SELCOPY executable.

The parameter string comprises one or more character constants that each generates a parameter variable equate in the executing SELCOPY program.

If the length of the parameter string is less than 80, the string is padded with blanks up to 80 characters and the null terminator (x'00') immediately follows the 80th character. If the string is longer than 80, the null terminator occurs immediately following the last character of the parameter string.

In the following invocation of SELCOPY...

```
SELCOPY 'Parm 1' 'Parameter2'  -ctl /home/xuser/ssparm01  -lst /tmp/ssparm01.lst
```

...the contents of the field at POS PARM are:

```
'Parm 1' 'Parameter2'     <59 trailing blanks follow>
```

See also internal field ARG which contains not only the parameter string but also any command line control statements, control statement input file and report output file specification that occurs on the SELCOPY invocation.

RBA

POS RBA is the position of a 4-byte binary field containing the junior 4 bytes of an 8-byte relative byte address (RBA), i.e. RBA-4 is the position of the full 8-byte binary field. Note that, an RBA is the offset of a particular record from the start of the file to which it belongs.

The RBA field will contain the offset of a record belonging to the last file processed by SELCOPY. For input files, the RBA will be that of the last record read. For output files, the RBA will be that of the next output record to be written, not the last record written.

For file sizes in excess of 4G, the senior 4 bytes of the RBA at position RBA-4 will be non-zero.

On input, the RBA field may be used to save the current record's RBA before proceeding to read further records form the same input file. The saved RBA may be used later to re-read the record directly using a READ by RBA operation.

SCALE

POS SCALE is the position of a character field of fixed length equal to the value of the DATAWIDTH option. The SCALE field contains the scale used in the footer line displayed below printed text in the print block of the SELCOPY list output.

UXADIFF

POS UXADIFF is the position of a 4-byte binary field containing a storage address of the position referenced by internal variable, DIFF (i.e POS DIFF). If DIFF is unset (DIFF = NULL), the UXADIFF field contains zeros (0).

The value assigned to DIFF and the UXADIFF field are both set automatically following execution of an IF character compare operation on a single field location. Furthermore, any direct assignment of variable, DIFF, will also update the contents of the UXADIFF field. Note that DIFF and UXADIFF are unchanged for a character compare operation on multiple field locations (i.e. an IF range test).

If no differences are found in a character compare operation on a single field location, the DIFF value and UXADIFF field are both unset, i.e. NULL and reset to zero (0). Otherwise, they reference the position of the first unmatched character within the first field (or character variable) element of the compare operation.

The POS DIFF address in the UXADIFF field is in big endian format, regardless of the underlying processor architecture on which SELCOPY is running. It is equivalent to the base address plus the value of DIFF minus 1.

UXATPTR

POS UXATPTR is the position of a 4-byte binary field containing a storage address of the position referenced by the default @variable, @ (i.e. POS @). If @ is unset (@ = NULL), the UXADIFF field contains zeros (0).

The value assigned to @ and the UXATPTR field are both set automatically following execution of an IF character compare operation on multiple field locations (i.e. an IF range test), where no PTR (or PTR=@) has been specified. Furthermore, any direct assignment of @variable, @, will also update the contents of the UXATPTR field.

The UXATPTR address field may also be updated directly with a 4-byte, big endian format storage address in order to point POS @ at that location. An advanced use of this feature would be to chain through z/OS system control blocks (see *"IBM z/OS*

MVS Data Areas" manuals) in order to obtain information about the environment in which SELCOPY is executing. e.g. To obtain the RACF userid assigned to the current task...

```
pos uxatptr = x'0000,0010' !print from @ ty=d l=016    * @-> Address of CVT.
pos uxatptr = 4 at @       !print from @ ty=d l=016    * @-> CVT.
pos uxatptr = 4 at @       !print from @ ty=d l=016    * @-> TCB Ptrs.
pos uxatptr = 4 at @+012   !print from @ ty=d l=112    * @-> ASCB.
pos uxatptr = 4 at @+108   !print from @ ty=d l=208    * @-> ASXB.
print from @+192 l=8                                   *     ASXBUSER.
pos uxatptr = 4 at @+200   !print from @ ty=d l=032    * @-> ACEE.
print from @+021 l=8                  * RACF userid.   *     ACEEUSRI.
```

If no matches are found by a character compare range test operation with no PTR parameter or with PTR=@ specified, the @ value and UXATPTR field are both unset, i.e. NULL and reset to zero (0). Otherwise, they reference the position of the first character of the first matching field within the range of fields tested.

The POS @ address in the UXATPTR field is in big endian format, regardless of the underlying processor architecture on which SELCOPY is running. It is equivalent to the base address plus the value of @ minus 1.

UXDW

POS UXDW is the position of a 4-byte binary field containing the print output data width value as defined by the DATAWIDTH option.

Note that data width is established during control statement analysis and cannot be updated during selection time processing. i.e. The data width value is not changed by an update to the UXDW field value.

UXINCNT

POS UXINCNT is the position of a 4-byte binary field containing the count of records read from the prime input data object.

The UXINCNT field value is equal to the value of internal variable, INCOUNT for the prime input data object. Note that INCOUNT values are maintained by SELCOPY and cannot be updated by a variable assignment statement. Similarly, the INCOUNT value for the prime input data object is not changed by an update to the UXINCNT field value.

For READ DIRDATA input, a separate count is maintained for input records of type directory and data. If the prime input data object is read with DIRDATA, then both the INCOUNT variable and UXINCNT field values reflect the record count of the type of record (directory or data) last read from the prime input data object.

If prime input comprises a number of data objects, read using the DIRDATA option and/or concatenated using the CAT sub-operation, then UXINCNT is reset to 1 following input of the first data record from each of the data objects.

Synonym: **UXINCOUNT**

UXLINE

POS UXLINE is the position of a 4-byte binary field containing the line number of the next record to be written to the SELCOPY list output. The line number value is the line number within the current page of output.

The maximum number of lines in each page of list output is defined by the PAGEDEPTH option and may be interrogated at selection time using the UXPD field. The number of lines remaining in the current page of output may be obtained from the UXLINEREM field.

The UXLINE field value is equal to the value of internal variable, LINE. The value of LINE may be updated via a variable assignment statement during selection time processing. Doing this will also update the UXLINE field value and the next SELCOPY list output record will either insert blank lines so that the record is written at the new line number, or throw a new page so that the record is written on the first available line following the page header lines. Note, however, that the LINE value is not changed by an update to the UXLINE field value.

UXLINEREM

POS UXLINEREM is the position of a 4-byte binary field containing the number of lines remaining in the current page of the SELCOPY list output.

The maximum number of lines in each page of list output is defined by the PAGEDEPTH option and may be interrogated at selection time using the UXPD field. The page line number of the next record to be written to the list output may be obtained from the UXLINE field.

Note that page depth is established during control statement analysis and cannot be updated during selection time processing. i.e. The page depth value is not changed by an update to the UXLINEREM field value.

UXLRECL

POS UXLRECL is the position of a 4-byte binary field containing the value of the internal variable, LRECL.

The value assigned to LRECL and the UXLRECL field are both set automatically to be the length of the last input record read via the READ operation. Furthermore, any direct assignment of variable LRECL will also update the contents of the UXLRECL field. Note, however, that the LRECL value is not changed by an update to the UXLRECL field value.

UXPD

POS UXPD is the position of a 4-byte binary field containing the maximum number of lines in each page (page depth) of the SELCOPY list output, as defined by the PAGEDEPTH option.

Note that page depth is established during control statement analysis and cannot be updated during selection time processing. i.e. The page depth value is not changed by an update to the UXPD field value.

UXPGNO

POS UXPGNO is the position of a 4-byte packed decimal field containing the current page number of the SELCOPY list output.

UXPW

POS UXPW is the position of a 4-byte binary field containing the page width value as defined by the PAGEWIDTH option. The page width is applied to header lines written to the SELCOPY list output and also governs the width of PRINT TYPE=D output.

Note that page width is established during control statement analysis and cannot be updated during selection time processing. i.e. The page width value is not changed by an update to the UXPW field value.

UXREASCD

Applicable only to z/VM CMS, POS UXREASCD is the position of a 4-byte binary field containing the value of the internal variable, REASCD, the reason code returned by a CMS function. See also UXRETSYS which contains the return code returned by a CMS function.

The value assigned to REASCD and the UXREASCD field are both set automatically following execution of a CMS function. e.g. The UTIME operation involves execution of a CMS function that sets the value assigned to REASCD and the UXREASCD field. Any direct assignment of variable REASCD will also update the contents of the UXREASCD field, however, the REASCD value is not changed by an update to the UXREASCD field value.

UXREPLYL

POS UXREPLYL is the position of a 4-byte binary field containing the length of text entered by the user following the last execution of a LOG operation with parameter REPLY.

For any particular LOG operation on which REPLY is specified, the value in the UXREPLYL field may not exceed the length of the REPLY field.

UXRETCD

POS UXRETCD is the position of a 4-byte binary field containing the value of the internal variable RETCD, SELCOPY's current return code. By default, the current SELCOPY return code is the highest return code value set by SELCOPY at any particular time during the program execution.

The value assigned to RETCD and the UXRETCD field are both set automatically by SELCOPY if a return code condition is triggered and the return code value exceeds the highest value set so far during the program execution.

Any direct assignment of variable RETCD will also update the contents of the UXRETCD field, however, the RETCD value is not changed by an update to the UXRETCD field value. Note that the return code may be updated to be any value as a result of a direct assignment on variable RETCD, regardless of whether it exceeds the current return code value.

UXRETSYS

POS UXRETSYS is the position of a 4-byte binary field containing the value of the internal variable RETSYS. This is the last return code set by the system following a command or function initiated by SELCOPY during the course of the program execution. This includes commands executed via the CP and SYSTEM operations and system functions performed by the UTIME and XV operations.

The value assigned to RETSYS and the UXRETSYS field are both set automatically following execution of a system command or function. Any direct assignment of variable RETSYS will also update the contents of the UXRETSYS field, however, the RETSYS value is not changed by an update to the UXRETSYS field value.

VOLID

POS VOLID is the position of a fixed length 32 character field which, for Microsoft Windows contains the volume label of the disk to which the last file processed by SELCOPY belongs. On iSeries, Linux and Unix operating systems, this field contains the host name (node name) of the system on which SELCOPY is running.

The volume label or host name value is either truncated at 32 characters or padded with blanks. SELCOPY return Code 8 is set (see RETCD and UXRETCD) if the VOLID field is referenced by an operation and the last file processed is not a local or networked disk file. e.g. READ STDIN and WRITE STDOUT.

Data Types

Constants, variables and field definitions are data elements that have an explicit or implied data type. For all constants, variables and fields, the data type is the format in which the assigned value is held in storage.

See Internal Field Definitions for the data types of SELCOPY's internal fields.

Data types may be split into 3 categories: Character, Date and Numeric.

Character Data Types

Character data types are interpreted as printable ASCII or EBCDIC character text.

By default, character text is interpreted as being in the encoding format (ASCII or EBCDIC) as defined by the machine architecture on which SELCOPY executes. Where applicable, this may be overridden on individual SELCOPY operations using keyword identifiers ASC (ASCII) or EBC (EBCDIC) as appropriate.

Character constants, variables and field definitions may be referenced within a character compare operation and also support position offsets within the assigned character text value.

The character data types recognised and supported by SELCOPY and the data elements to which they apply, are discussed below.

Character Fixed Length

A text string value of fixed length.

Constants:
Character constants are all of fixed length and their specification is discussed earlier in this chapter.

Variables:
Declared variables of this data type are declared using keyword identifier CHAR, with optional length value enclosed in parentheses delimiter characters. e.g.

```
DECLARE   POSTCODE   CHAR(10)         * Character fixed length 10.
```

Fields:
A field is at a position and length defined by the field definition syntax specified on the operations in which it is referenced. The field definition syntax includes data type specification, TYPE=C, to indicate character data type. For MOVE, MOD and IF character compare operations, TYPE=C is default. e.g.

```
MOD  POS 11  LENGTH 5  TYPE=C  =  '1'  FILL='1'    * Value:   11111
ADD    12345    TO   5 AT 11  TYPE=C                * Becomes: 23456
```

Character Variable Length

A text string value of variable length.

The source field for this data type has a length equal to the maximum defined length and a 2-byte binary field prefix containing only the length of the value text. Residual text occupying the area of the source field beyond the current value's length, is unchanged.

Variables:
Only declared variables may be defined with this data type. These may be declared using keyword identifier VARCHAR, with an optional maximum length value enclosed in parentheses delimiter characters. e.g.

```
DECLARE   TITLE    VARCHAR(64)        * Character variable, max length 64.
DECLARE   VVAR     VCH(100)  INI 'XY'

PRINT    'VVAR is:-->' VVAR '<--'    * Gives: "VVAR is:-->XY<--"
VVAR = 'ABCDEF'
PRINT    'VVAR is:-->' VVAR '<--'    * Gives: "VVAR is:-->ABCDEF<--"
```

Character Varying Length

Character varying length data type is identical to the character variable length data type except that each character of residual text occupying the area of the source field beyond the current value's length, is replaced with the prevailing pad character (OPTION FILL). The default pad character is blank.

This data type matches the PL/1 definition of data items declared with the CHARACTER VARYING attribute. Data written by PL/1 based on this data type will have the 2-byte binary prefix followed by the character field of maximum text length.

Variables:
Only declared variables may be defined with this data type. These may be declared using keyword identifier CHARVARYING, with an optional maximum length value enclosed in parentheses delimiter characters. e.g.

```
DECLARE   VNAME   CHARVARING(32)   * Character variable, max length 32.
```

Character Variable Null Terminated

A text string value of variable length terminated by the null character (x'00').

The source field for this data type has a length equal to the maximum defined length plus 1 for the null terminator. This data type matches the PL/1 definition of data items declared with the CHARACTER VARYINGZ attribute.

Variables:
Only declared variables may be defined with this data type. These may be declared using keyword identifier NTS, with an optional maximum length value enclosed in parentheses delimiter characters. e.g.

```
DECLARE   STRCHR   NTS(100)        * Character null terminated max length 100.
```

Date Data Types

Date data types identify data interpreted as a chronological date and are currently supported by the CVDATE operation only.

Date constants may be specified as a CVDATE source value only, whereas date field definitions are valid as either the source or target of the CVDATE operation.

Date data types are not supported natively for declared variables. However, a variable of source data type character, binary or decimal (scale=0) may be used with a STYLE or FORMAT specification in place of a field definition of TYPE=C, TYPE=B or TYPE=P respectively.

Each of the date data types supported by SELCOPY, may be used to represent the different supported styles of date format. Specifically, these are:

STYLE Option	Date Format	Digits
A	American (USA)	`mmddccyy or mmddyy (1)`
B	British/European	`ddmmccyy or ddmmyy (1)`
D	Number of days starting 1900/01/01	`+n or -n`
I	International Standard (ISO)	`ccyymmdd or yymmdd (1)`
J	Julian	`ccyyddd or yyddd (1)`
T	z/Architecture TOD clock	`X'xxxx,xxxx,xxxx,xxxx'`

Legend:

cc	2-digit century number.
yy	2-digit year of century number.
mm	2-digit month of year number.
dd	2-digit day of month number.
ddd	3-digit day of year number.
n	decimal integer value.
x	hexadecimal digit.

Notes:

1. ISO, American, European and Julian format dates support source date values with or without a 2-digit century. However, CVDATE output of any of these date formats, will always contain a 4-digit year. Return code 8 is set if the target field is too small.

   A source date value without a century will have an implied century of 20 for years 00-49, or 19 for years 50-99.

Date data types recognised and supported by SELCOPY and the data elements to which they apply, are discussed below.

Character Date

Character (TYPE=C) data may have a date interpretation when referenced as the source or target of a CVDATE operation.

If specified as the **target** of a CVDATE operation, a character field definition or variable may be formatted using a DATE FORMAT string. If no formatting is performed, the target character date value will contain only numerical digits and no punctuation. e.g. '2011/03/15' (STYLE=I) would be left-adjusted in the target field as '20110315'.

If specified as the **source** of a CVDATE operation, the date value represented by a character constant or field definition is dependent on the specified STYLE.

Character Date Source

In character date source values, numeric characters (0-9) correspond to decimal numeric digits (0-9) respectively and all non-numeric characters are ignored. Therefore, non-numeric characters may occur at any position within the character source (e.g. as year/month/day delimiters) without affecting the date value.

Each style of date has a valid number of decimal numeric digits that must be specified. If the number of specified digits are invalid, then a control statement error will occur for date character constants, otherwise return code 8 is set for the CVDATE operation and no date conversion is attempted.

The date value is interpreted by evaluating the character data from right to left (i.e. highest to lowest storage address) and establishing values for the individual date components (e.g. year, month, day) that constitute the date style specified.

Having identified the individual date components, return code 8 is set and date interpretation is cancelled if a component does not satisfy the component limits . i.e. month number must be in the range 1-12, day of month number in the range 1-28/29/30/31 (depending on the month) and day of year number in the range 1-365/366 (depending on the year.)

STYLE=A and STYLE=B
STYLE=A (USA format) and STYLE=B (British/European format) character dates must contain either 6 or 8 numeric digits.

If the character date has 6 digits, SELCOPY treats the right-most 2 digits as a year specification with an implied century of 20 for years 00-49, or 19 for years 50-99. If the date has 8 digits, SELCOPY treats the right-most 4-digits as the year specification with explicit century number.

Thereafter, the next 2 pairs of digits processed from right to left correspond to the day then numbers (STYLE=A), or month then day numbers (STYLE=B).

STYLE=D
STYLE=D character dates must include at least 1 numeric digit representing the day number starting at 1900/01/01.

STYLE=I (default)
STYLE=I (ISO format) character dates must contain either 6 or 8 numeric digits.

SELCOPY treats the right-most 2 digits as the day number and the next 2 digits as the month number. If the character date has 6 digits, SELCOPY treats the next 2 digits as the year specification with an implied century as described for STYLE=A and B. Otherwise, the next 4-digits constitute the year specification with explicit century number.

STYLE=J
STYLE=J (Julian format) character dates must contain either 5 or 7 numeric digits.

SELCOPY treats the right-most 3-digits as the day of year number. If the character date has 5 digits, SELCOPY treats the next 2 digits as the year specification with an implied century as described for STYLE=A and B. Otherwise, the next 4-digits constitute the year specification with explicit century number.

STYLE=T
Dates of STYLE=T (TOD clock) are of TYPE=B (8-byte binary) and so no character date interpretation is performed.

Examples of source date character values:

Character Data (Quoted Constant)	Date Value (FORMAT='yyyy/mm/dd')
'2012/031' STYLE=J	2012/01/31
'12/11/71' STYLE=B	1971/11/12
'40,789' STYLE=D	2011/09/04
'"06","17","07"' STYLE=A	2007/06/17
'02#bb11#bbb1983' STYLE=A	1983/02/11
'200/80/703' STYLE=I	2008/07/03
'315' STYLE=D	1900/11/11

Constants:

Character date constant elements are all of fixed length and their specification is discussed earlier in this chapter.

Fields:

The field definition syntax includes data type specification, TYPE=C, to indicate character data type and a date STYLE specification. TYPE=C, STYLE=I is default. e.g.

```
CVDATE         '2014/07/12'           TO 1 FORMAT='yyyy/ddd' * Value: 2014/193
CVDATE POS 1 LENGTH 8 TYPE=C STYLE=J  TO 6 AT 11 STYLE=D     * Value: 041831
```

Binary Date

A TYPE=B binary (base-2) integer value may be interpreted as a date when referenced as the source or target of a CVDATE operation. This integer value is a big-endian, binary representation of the decimal digits that constitute a date value.

For any given date, the value will be different depending on its style attribute. e.g. ISO date, 2015/08/25, has a decimal value 20150825 (STYLE=I) or 2015237 (STYLE=J). Represented as hex constants, these decimal values have binary equivalents x'0133,7A29' (STYLE=I) and x'001E,C005' (STYLE=J) respectively.

Note that a source binary integer that does not represent a valid date of the style specified will cause the CVDATE operation to fail with return code 8.

Binary date values of STYLE=D are of practical use when date arithmetic is required. e.g. A character format date may be converted to binary STYLE=D before adding or subtracting a number of days then converting it back to character display format. A similar method may be used to subtract one date from another in order to determine a number of elapsed days.

A date value of STYLE=T is a binary value of length 8-bytes. Specification of TYPE on a STYLE=T date value is ignored.

Constants:

Binary date constants are represented as hex character constants with TYPE=B and STYLE parameters. Their specification is discussed under Date Character Constants.

Fields:

Binary date field definition syntax includes data type specification, TYPE=B and a date STYLE specification. e.g.

```
CVDATE  4 AT 1 TYPE=B STYLE=J  TO  11 FORMAT='yyyy-mm-dd'   * Binary Julian.
CVDATE          NOW            TO  4 AT 31 TYPE=B STYLE=D   * Binary #days.
CVDATE  8 AT 101      STYLE=T  TO  4 AT 41 TYPE=B STYLE=I   * Binary TOD->ISO.
```

Unsigned Decimal Date

TYPE=U unsigned packed decimal integer values are supported only as a date value specification on the CVDATE operation.

The decimal digits that constitute a date value may be in a packed, big endian format so that each digit of the decimal value is represented by a hex code (x'0'-x'9'). The style of date determines the number of digits in the packed value and so the length of the packed data.

Return code 8 is set for the CVDATE execution if the unsigned packed decimal value does not represent a valid date in the specified style.

Examples of unsigned packed decimal date values:

Unsigned Decimal (Hex Constant)	Date Value (FORMAT='yyyy/mm/dd')
X'02012031' TYPE=U STYLE=J	2012/01/31
X'030563' TYPE=U STYLE=B	1963/05/03
X'01221993' TYPE=U STYLE=A	1993/01/22
X'20131030' TYPE=U STYLE=I	2013/10/30
X'11213' TYPE=U STYLE=D	1930/09/13

Constants:
Unsigned decimal date constants are represented as hex character constants with TYPE=U and STYLE parameters. Their specification is discussed under Date Character Constants.

Fields:
Unsigned decimal date field definition syntax includes data type specification, TYPE=U and a date STYLE specification. e.g.

```
CVDATE '2015/08/25'   STYLE=I TO 4 AT 11 TYPE=U STYLE=A  * Value: X'0825,2015'
CVDATE  4 AT 1 TYPE=U STYLE=J TO   1 FORMAT='yyyy/ddd'   * Unsigned Julian.
```

Signed Decimal Date

A TYPE=P signed packed decimal integer value may be interpreted as a date when referenced as the source or target of a CVDATE operation.

The decimal digits that constitute a date value may be in a packed, signed, big endian format so that each digit of the decimal value is represented by a hex code (x'0'-x'9') and the value's sign is represented by a hex code (x'A'-x'F') in the least significant position. Sign codes x'A', x'C', x'E' and x'F' indicate a positive decimal value (+) whereas codes x'B' and x'D' indicate a negative value (-).

The style of date determines the number of digits in the packed value and so the length of the signed, packed data. Return code 8 is set for the CVDATE execution if the signed packed decimal value does not represent a valid date in the specified style.

Signed packed decimal dates values of STYLE=D are useful when date arithmetic is required. e.g. A character format date may be converted to signed packed decimal STYLE=D before adding or subtracting a number of days then converting it back to character display format.

Examples of signed packed decimal date values:

Signed Decimal (Hex Constant)	Date Value (FORMAT='yyyy/mm/dd')
X'2012,131C' TYPE=P STYLE=J	2012/05/10
X'0300,908C' TYPE=P STYLE=B	2008/09/30
X'0122D' TYPE=P STYLE=D	1899/08/31
X'0C' TYPE=P STYLE=D	1899/12/31
Note that x'C' and x'D' are SELCOPY's preferred representation of positive and negative sign codes, and are used in a TYPE=P target value of a CVDATE operation.	

Constants:
Signed decimal date constants are represented as hex character constants with TYPE=P and STYLE parameters. Their specification is discussed under Date Character Constants.

Fields:
Signed decimal date field definition syntax includes data type specification, TYPE=P and a date STYLE specification. e.g.

```
CVDATE '1999/12/13'    STYLE=I  TO  8 AT 11 TYPE=P STYLE=D   * PD #days.
ADD         '22'                TO  8 AT 11 TYPE=P           * +22 days.
CVDATE  8 AT 11 TYPE=P STYLE=D  TO  1 FORMAT='yyyy/mm/dd'    * Value: 2000/01/04
```

Numeric Data Types

Numeric data types identify computational data.

Constants, variables and field definitions that are numeric, or are character with a numeric interpretation, may be used in arithmetic and arithmetic compare operations.

Numeric data types recognised and supported by SELCOPY and the data elements to which they apply, are discussed below.

Binary Integer

A binary (base-2) integer value.

The precision (number of binary digits) that may be attributed to a binary integer value and whether or not it is signed or unsigned, is determined by the length of the source field. A binary integer source field may be between 1 and 4-bytes in length with the following implications:

Field Length	Signed/Unsigned	Maximum Precision
1	Unsigned	8
2	Signed	15
3	Signed	23
4	Signed	31

Regardless of the processor architecture on which SELCOPY is running, hex representation of binary integer source fields is in big-endian format. e.g.

```
DECLARE       BINVAR  BIN(3)   INI=100
PRINT   "x'" &BINVAR  FMT="xx,"  "'"      * Prints x'00,00,64', not x'64,00,00'
```

Constants:

Binary integer constants are represented as hexadecimal values their specification is discussed earlier in this chapter.

Variables:

Declared variables of this data type are declared using keyword identifier BIN, with optional length value enclosed in parentheses delimiter characters. e.g.

```
DECLARE   TOTALS          BIN(4)     * Signed binary integer, precision 31.
```

Fields:

Binary integer field definition syntax includes data type specification, TYPE=B. For CVBx operations, TYPE=B is default. e.g.

```
IF   POS 33  LENGTH 4  TYPE=B   >   80        * Arithmetic compare.
 THEN SUB  12   FROM 4 AT 33  TYPE=B          * Subtract 12.
 THEN CVBC  33, 36   TO   101 FMT='s,ss9'     * Convert binary to character.
```

Decimal Integer

A signed decimal integer value.

A decimal integer value is stored in a packed decimal source field. Packed decimal values are supported by SELCOPY on all operating platforms and are always represented in big endian format. Each decimal digit of the decimal integer value is represented by a hex code (x'0'-x'9') and the value's sign is represented by a hex code (x'A'-x'F') in the least significant position of the field. Sign codes x'A', x'C', x'E' and x'F' indicate a positive decimal value (+) whereas codes x'B' and x'D' indicate a negative value (-).

The precision (number of decimal digits) that may be attributed to a decimal integer value is determined by the length of the source field. Since all but 1 of the bytes in a signed packed decimal field can represent 2 decimal digits, the maximum precision of a decimal value is (2x*Field_Length*)-1, where 1 <= *Field_Length* <= 16.

Source Field	Maximum Precision	Decimal Value
x'1C'	1	+1
x'00001C'	5	+1
x'0003426F'	7	+3426
x'0003426D'	7	-3426
Note that x'C' and x'D' are SELCOPY's preferred representation of positive and negative sign codes, and are used in the target field of a CVxP operation.		

Constants:

Specification of decimal integer constants is discussed earlier in this chapter.

Variables:

Declared variables of this data type are declared using keyword identifier DEC, with an optional precision value (but no scale value) enclosed in parentheses delimiter characters. e.g.

```
DECLARE   FILE_COUNT   DEC(7)    * Decimal integer, precision 7 (field length 4).
```

The packed decimal source field will occupy the minimum length required to accommodate the variable's precision.

Fields:

Packed decimal field definition syntax includes data type specification, TYPE=P. For CVPx and arithmetic operations (ADD, SUB, MULT and DIV), TYPE=P is default. e.g.

```
IF   POS  1  LENGTH 8  TYPE=P   <>   23        * Arithmetic compare.
 THEN ADD  12             TO  8 AT 1          * Add 12. (Default TYPE=P)
 THEN DIV   8 AT 1 TYPE=P  BY  4 AT 33 TYPE=B  * Divide by binary field value.
 THEN CVPC  1, 8   TO    81 FMT='s,ss9'      * Convert packed decimal to char.
```

Zoned Decimal Integer

A signed zoned decimal integer value.

A zoned decimal integer value is stored in a zoned decimal source field. Zoned decimal values are supported by SELCOPY on all operating platforms and are always represented in EBCDIC, **big endian** format. If the base code page encoding scheme used by the local system is ASCII, a zoned decimal source field referenced by a SELCOPY operation will be converted to EBCDIC before being processed, and then converted back to ASCII if the value has been updated.

Each digit of the zoned decimal integer value is represented by one byte in the EBCDIC source field. The right-most 4 bits of the byte denote the number which is the digit value represented as a hex code (x'0'-x'9').

For all digits except the least significant (right-most) digit, the left-most 4 bits of the byte denote the zone. The zone bits contain a fixed hex code (x'F') so that the zone and number bits together constitute a printable EBCDIC character value corresponding to the decimal digit. i.e. X'F0' to x'F9' is EBCDIC characters 0 to 9.

The left-most 4 bits of the least significant byte of the EBCDIC source field, denote the value's sign. Like packed decimal, sign is represented by a hex code (x'A'-x'F') with codes x'A', x'C', x'E' and x'F' indicating a positive decimal value (+) and codes x'B' and x'D' indicating a negative value (-). Note, however, that SELCOPY only supports zoned decimal values with the industry preferred sign codes: x'C', x'D' or x'F'.

The precision (number of decimal digits) that may be attributed to a decimal integer value is equal to the length of the source field.

Character Print	Decimal Value	Source (EBCDIC)	Source (ASCII)
1234	+1234	x'F1F2F3F4'	x'31323334'
95C	+953	x'F9F5C3'	x'393543'
123M	-1234	x'F1F2F3D4'	x'3132334D'
3J	-31	x'F3D1'	x'314A'
4 Note that x'F' and x'D' are SELCOPY's preferred representation of positive and negative zoned decimal sign codes, and are used in the target field of a CVxZ operation.			

Constants:
Specification of zoned decimal integer constants is discussed earlier in this chapter.

Fields:
A zoned decimal integer field definition must include data type specification, TYPE=Z. Alternatively, for field definitions that are not Type 4, TYPE=C may be used. e.g.

```
ADD   4 AT 1 TYPE=B  TO  8 AT 21 TYPE=Z   * Add binary integer to a zoned int.
CVZC  21, 28   TO   31 FMT='£ s,ss9 DR'   * Convert zoned to formatted char.
```

Decimal Fixed Point

A signed decimal fixed point (rational number) value.

A decimal fixed point value has both a precision and scale (p,s) and is stored in a packed decimal source field as described for decimal integer values. The precision identifies the maximum number of decimal digits that may be attributed to the decimal fixed point value, whereas scale identifies the number of those digits that are reserved for the fraction. e.g. A value 123.45 has precision and scale of (5,2).

As for any decimal number representation, the fraction digits occupy the least significant positions. For decimal fixed point values, these correspond to the least significant positions of the packed decimal source field before the sign code.

Constants:
Specification of decimal fixed point constants is discussed earlier in this chapter.

Variables:
Declared variables of this data type are declared using keyword identifier DEC, with precision and scale value enclosed in parentheses delimiter characters.

```
DECLARE   AREA    DEC(6,2)    * Decimal precision 6, scale 2 (field length 4).
```

The packed decimal source field will occupy the minimum length required to accommodate the variable's precision.

Fields:
Packed decimal field definition syntax does not support fixed point specification.

Zoned Decimal Fixed Point

A signed zoned decimal fixed point (rational number) value.

SELCOPY supports interpretation of a decimal point within zoned decimal source fields to denote integer and fraction digits within the zoned decimal value. (e.g. 8.1J) See zoned decimal integer for description of zoned decimal field contents.

A zoned decimal fixed point value has both a precision and scale (p,s). The precision identifies the maximum number of zoned decimal digits that may be attributed to the decimal fixed point value, whereas scale identifies the number of those digits that are reserved for the fraction. e.g. A value 123.4E has precision and scale of (5,2).

As for any decimal number representation, the fraction digits occupy the least significant positions. For zoned decimal fixed point values, these correspond to the least significant positions of the zoned decimal source field before the sign code.

If a zoned decimal fixed point value is used in SELCOPY syntax where only an integer value is appropriate, then the fraction digits are ignored. e.g. when assigning a value to an integer type variable.

Constants:

Specification of zoned decimal fixed point constants is discussed earlier in this chapter and includes examples of zoned decimal fixed point values.

Fields:

Zoned decimal fixed point field definitions may only be used as source fields in SELCOPY operations and variable assignments. If specified as an operation target field, the decimal point is ignored and the value treated as a zoned decimal integer value.

A zoned decimal fixed point field definition must include data type specification, TYPE=Z. Alternatively, for field definitions that are not Type 4, TYPE=C may be used. e.g.

```
DECLARE   B1  BIN                          * Binary integer variable, B1.
DECLARE   F1  FLT(8)  BIN                  * Floating point variable, F1.
MOD           21  =  '3123.4M'             * Value: -3123.44
B1   =   7 AT 21  TYPE=Z                   * B1 = -3123
F1   =   7 AT 21  TYPE=Z                   * F1 = -3123.44
SUB   4 AT 1 TYPE=Z  FROM    F1            * Subtract a rational number.
CVZC  21, 27  TO  31 FMT='£s,ss9.99 DR'    * '£3,123.44 DR'
```

Hex Floating Point

SELCOPY supports rational numeric values sourced from finite, single precision (short format: 4-byte) or double precision (long format: 8-byte) hexadecimal floating point fields. Extended format (16-byte) floating point fields are not supported.

A hex floating point field comprises a sign bit, exponent and significand (mantissa) to represent rational hexadecimal numbers where the position of the radix point can vary.

The most significant bit determines the sign (0=Positive, 1=Negative). An unsigned binary value, defined by the next 7 bits, includes a x'40' bias and represents the signed, base-16 exponent value (i.e. powers of the radix 16). The remaining bits of the field represent the significand, a 6 hex digit (single precision) or 14 hex digit (double precision) value. The significand value is a fraction since the implied radix point is always to the left of this value.

Unless the DEFAULTFP environment option has been set to BIN, hex floating point is SELCOPY's default floating point format when running on IBM z/OS or z/VM CMS.

Constants:

Floating point values comprising a sign, significand and exponent may be specified as source data on a CVCF operation only. This is discussed under Numeric Character Constants earlier in this chapter.

Variables:

Declared variables of this data type are declared using keyword identifier FLT or DBL with parameter HEX. Note that HEX is default for z/OS and z/VM CMS.

```
DECLARE   FS   FLT    HEX   * Single precision hex floating point.
DECLARE   FD   FLT(8) HEX   * Double precision hex floating point.
DECLARE   DD   DBL    HEX   * Double precision hex floating point.
```

Fields:

Hex floating point field definition syntax includes data type specification, TYPE=F, with parameter HEX. Note that HEX is default for z/OS and z/VM CMS. e.g.

```
CVCF 20 AT   1             TO   8 AT 101   TYPE=F HEX     * Char -> HEX Float.
CVFC  8 AT 101 TYPE=F HEX  TO  21   FORMAT='z,zz9.99999-' * HEX Float -> Char.

IF    8 AT 101 TYPE=F HEX       <=   1.9      * Arithmetic compare.
 THEN MULT   8 AT 101 TYPE=F HEX    BY  16    * Multiply by 16.
```

Specification of a hex floating point field with a length greater than 8 bytes will give return code 8 if the field represents the source value of a variable assignment, or ERROR 062 if a parameter to a SELCOPY operation.

However, specification of non-standard floating point field lengths (1, 2, 3, 5, 6, 7) are acceptable. For the purposes of value assignment, evaluation, arithmetic and data type conversion, floating point fields defined with one of these non-standard lengths are padded on the right with binary zeros to a length of 4 or 8 bytes as appropriate. If specified as the target field of a conversion operation (CVxF), the least significant bytes of the significand value will be truncated or padded with binary zeros to accommodate the target field length.

Binary Floating Point

SELCOPY supports rational numeric values sourced from finite, single precision (short format: 4-byte) or double precision (long format: 8-byte) IEEE-754 binary floating point fields. Extended format (16-byte) floating point fields are not supported.

A binary floating point field comprises a sign bit, exponent and significand (mantissa) to represent rational binary numbers where the position of the radix point can vary.

The most significant bit determines the sign (0=Positive, 1=Negative). Bits immediately following the sign bit, define a biased, unsigned binary value that represents a signed base-2 exponent value (i.e. powers of the radix 2). The number of bits that define the exponent value and the bias applied depend on the binary floating point field format.

Precision	#Bits	Bias
Single (short)	8	127 (x'7F')
Double (long)	11	1023 (x'3FF')

The remaining bits of the field represent the significand value, a 23 binary digit (single precision) or 52 binary digit (double precision) value. The significand value is a fraction since the implied radix point is always to the left of this value.

Normal binary floating point values have an implied integer value of 1 (i.e. the unit bit is 1). For sub-normal binary floating point values (exponent is zero, significand is non-zero), the implied integer value is 0. Note that sub-normal binary floating point fields are not supported by SELCOPY conversion operations (CVxx).

A binary floating point field where the exponent bits are all ones, represent **INFINITY** if the significand is zero or **NaN** (Not a Number) if the significand is non-zero.

Unless the DEFAULTFP environment option has been set to HEX, binary floating point is SELCOPY's default floating point format when running on operating systems other than IBM z/OS and z/VM CMS.

Constants:

Floating point values comprising a sign, significand and exponent may be specified as source data on a CVCF operation only. This is discussed under Numeric Character Constants earlier in this chapter.

Variables:

Declared variables of this data type are declared using keyword identifier FLT or DBL with parameter BIN.

```
DECLARE   FSB  FLT    BIN   * Single precision bin floating point.
DECLARE   FDB  FLT(8) BIN   * Double precision bin floating point.
DECLARE   DDB  DBL    BIN   * Double precision bin floating point.
```

Fields:

Binary floating point field definition syntax includes data type specification, TYPE=F, with parameter BIN. Note that BIN is default for Unix, Linux and Windows. e.g.

```
CVCF 20 AT   1             TO   4 AT 201   TYPE=F BIN     * Char -> BIN Float.
CVFC  4 AT 201 TYPE=F BIN  TO  21   FORMAT='z,zz9.99999-' * BIN Float -> Char.

IF    4 AT 201 TYPE=F BIN       >   -0.019    * Arithmetic compare.
 THEN MULT   4 AT 201 TYPE=F BIN    BY  24    * Multiply by 24.
```

Specification of a binary floating point field with a length greater than 8 bytes will give return code 8 if the field represents the source value of a variable assignment, or ERROR 062 if a parameter to a SELCOPY operation.

However, specification of non-standard floating point field lengths (2, 3, 5, 6, 7) are acceptable. For the purposes of value assignment, evaluation, arithmetic and data type conversion, floating point fields defined with one of these non-standard lengths are padded on the right with binary zeros to a length of 4 or 8 bytes as appropriate. If specified as the target field of a conversion operation (CVxF), the least significant bytes of the significand value will be truncated or padded with binary zeros to accommodate the target field length.

Numeric Character Data

Character data may have a numeric interpretation when referenced as one of the following:

- A source value for numeric variable assignment or arithmetic operations. (e.g. ADD, MULT)
- A formatted numeric display value for output operations. (e.g. PRINT, WRITE)
- A target field or declared variable for arithmetic or data conversion operations. (e.g. SUB, CVxC)

Declared character variables may only have a numeric interpretation if a FORMAT string is specified on the DECLARE operation. e.g.

```
DECLARE  C1   CHA(2)  INI='12'
DECLARE  C2   CHA     INI='12'   FMT='99'    * Numeric interpretation.

MOD  POS C1+1  =  'X'    * Value of C1              becomes:  '1X'
MOD  POS C2+1  =  'X'    * Value of field at pos 13 becomes:  'X'
```

Character constants and field definitions may have a numeric interpretation whether or not a FORMAT string is specified.

Numeric Character with FORMAT

If character data has an associated FORMAT string, then this will define the template by which the numeric value is established.

For target variables and fields, the FORMAT string also defines the template used to write numeric character data in storage. If FORMAT is specified on an output operation source value, an implicit CVCC operation is performed first to convert the source character value into the character display value specified by the FORMAT string template. This character display value is written to the output object. e.g.

```
POS    101  = 'Input Value: 1223.50'                    * Source field.
PRINT  'Rounded Value = '  FROM 20 AT 101 FMT='zz,zz9'   * Printed as: 1,224
```

See Format Specification Parameters for details on each FORMAT string symbol used for numeric value interpretation.

Numeric Character without FORMAT

If no FORMAT string has been provided, the numeric value represented by a character constant or field definition is interpreted by evaluating the character data from left to right (i.e. lowest to highest storage address) as follows:

1. If the last character is one of "C", "D" or "F" (upper case) and the penultimate character is either a numeric (0-9) or a decimal point (.), then the value is interpreted as being a zoned decimal integer or zoned decimal fixed point value.
2. If the value is not zoned decimal, a leading unary plus (+) or minus (-), specified as the first non-blank character, determines the sign of the value as positive or negative respectively. For zoned decimal values, the leading plus or minus is ignored.
3. Numeric characters (0-9) correspond to decimal numeric digits (0-9) respectively.
4. Not applicable to zoned decimal values, if no leading sign is specified but a non-numeric character, other than a decimal point (.), is specified to the right of the last numeric digit, then the sign of the value is negative.
5. The first occurrence of a decimal point character (.) determines the location of the decimal point. Numeric digits that follow this constitute the fractional portion of the number.
6. If used as source data on a **CVCF** (convert character to floating point) operation or on a floating point variable assignment, the first occurrence of the character 'e' or 'E' denotes the start of a signed integer exponent value. If the first character that follows is not minus (-), then a positive exponent is assumed. If no numerical digit exists in the exponent value, an exponent of zero (0) is assumed.
7. With exceptions as specified in previous points, all non-numeric characters are ignored.

Examples of numeric character values which have no associated FORMAT string:

Character Data	Numeric Value
2,675.1	+2675.1
2,675.1 CR	-2675.1
1K	-12
1k	-1
-A1AA8.AA,,2.1A4	-18.214
A-12.4	+12.4
3,dF1.h8L	-31.83
1.065e6	+1065000.0
-231.065e-2	-2.31065.0

FORMAT Strings

FORMAT strings are used to define the character display format of output values. e.g. field definitions or variables specified as the target of a CVxC operation or specified as values for output operations.

A FORMAT string is a sequence of FORMAT symbol characters of up to a maximum length 66, which describe the contents of each position of a numeric character data item and the contents of the character display output. It may be specified as an unquoted literal, quoted character constant or, for CVxC or CVxH target values only, as a field definition or declared variable.

SELCOPY supports the following 3 types of FORMAT string, each supporting a distinct set of FORMAT symbol characters.

- Numeric FORMAT strings.
- Printable Hex FORMAT strings.
- DATE FORMAT strings.

The type of FORMAT string is determined by the SELCOPY operation and the specified FORMAT symbols.

Numeric FORMAT Symbols

Numeric FORMAT strings may be used as follows:

1. To represent the unpacked, decimal character display format of numeric values. A character display format is applicable only for values that are the target (output) of an operation. This includes values that are specified on an output (PRINT, PLOG, WRITE) operation. e.g.

```
DECLARE   XXXVAR1 CHA(20)                * Character variable.
DECLARE   FLTVAR  FLT                    * Floating Point.

ADD  '12.3'   TO    4 AT 21 TYPE=F    INTO  101    FORMAT='--,--9.999'
CVPC        8 AT 12    TO    XXXVAR1 FORMAT='z,zzz,zz9 CR'
WRITE  "Variable FLTVAR= "   FLTVAR  FORMAT='s,sss,ss9.999'
PRINT  TYPE=C   FROM 4 AT 11 TYPE=B  FORMAT='S99.99'
```

2. To attribute a numeric interpretation to character declared variables and field definitions. The same FORMAT string also defines the default character display format for the variable or field. e.g.

```
DECLARE   XXXVAR2   CHA   FORMAT='ss,ss9'  * Character numeric variable.

XXXVAR2                          =  3247   * '+3,247'
MOD   POS 31  FORMAT='ss,ss9.99' = '-27.3' * '   -27.30'
PRINT  "Variable XXXVAR2="   XXXVAR2       * Uses FORMAT on DECLARE.
```

Numeric FORMAT strings contain characters that each represents a FORMAT **substitution control symbol**, a **decimal point control symbol** or a **constant value**. FORMAT substitution control symbols are digit, zero suppression and floating sign position control symbols.

Integer, fixed point and floating point source values may be represented by numeric integer or fixed point FORMAT strings. For integer FORMAT strings, no decimal point control symbol is specified and all substitution control symbols represent integer digit positions. For fixed point FORMAT strings, integer and fractional digit positions correspond to FORMAT substitution control symbols located to the left and right of the decimal point control symbol respectively.

The length of a numeric FORMAT string may not exceed 255 characters and must include at least as many FORMAT substitution control symbols representing **integer** digits as there are decimal integer digits in the numerical value. If a floating sign symbol is specified as the first FORMAT substitution control symbol, then the minimum number of substitution control symbols is increased by 1.

If the FORMAT string does not satisfy these criteria, then return code 8 (RETCODE=8) is flagged for the SELCOPY statement on which the error occurred and the character display format of the number is all asterisks (*) for the length of the FORMAT string.

Digit Control Symbol

The digit control symbol, "9" is a FORMAT substitution control symbol used to represent the position of a decimal digit (0-9) in the character display.

Digit control symbols are substituted with a numeric digit regardless of whether the digit is significant or insignificant within the numeric value. Insignificant digits are leading zeros that occur to the left of the first non-zero digit belonging to the integer portion of the source numerical value.

Examples of digit control symbols:

Source Value (as decimal)	FORMAT String	Character Display Format
12	99	12
12	999	012
143	99999	00143
2797	99999	02797
659	99	**

Decimal Point Control Symbol

The decimal point control symbol "." (dot/period) is used to identify the start of fraction digits in the character display of a fixed or floating point number. It is the position at which the decimal point character "." is situated in the character display.

If more than one decimal point control symbol is specified, a selection time error (ERROR 122) is returned. If no decimal point control symbol is specified in the FORMAT string, then the character display format is an integer.

The integer and fraction digits of the numeric source value are aligned on the decimal point control symbol. Therefore, the character display of a numeric value may be extended with insignificant zero digits before the integer value and following the fractional value.

For fixed point or floating point numeric values, if no decimal point control symbol is specified or there are fewer FORMAT substitution control symbols to the right of the decimal point control symbol than there are fractional digits, then rounding occurs

at the least significant digit control character.

Special Case:
Decimal point control symbol (".") is treated as being a character constant if it follows the last substitution symbol in the FORMAT string.

Examples of decimal point control symbol use:

Source Value (as decimal)	FORMAT String	Character Display Format
12	99.99	12.00
12.8	999.9	012.8
12.8	999	013
2.5	999.999	002.500
143.333	9999.99	0143.33
143.335	9999.99	0143.34
6659	99.99	*****

Zero Suppression Control Symbol

The zero suppression control symbols "z" and "Z" are FORMAT substitution control symbols used to represent the position of a conditional decimal digit (0-9) in the character display.

Zero suppression control symbols are interpreted as follow:

1. If the zero suppression symbol represents a digit which is a leading (i.e. insignificant) zero, then a blank character is substituted. Otherwise, the digit (0-9) is substituted as is.
   e.g. For value 22, `FMT='zzzz'` returns character display `bb22`.

2. A zero suppression symbol that occurs to the right of a digit control symbol ("9") is itself treated as a digit control symbol. i.e. no zero suppression will occur for these symbols.
   e.g. `FMT='zz9zz.zz'` is equivalent to: `FMT='zz999.99'`

3. If all substitution symbols in the FORMAT string are zero suppression symbols, then a zero integer, fixed point or floating point numeric value will have a character display of all blanks for the length of the FORMAT string.
   e.g. For value 0.00, `FMT='AzBzCD.zzEF'` returns character display `bbbbbbbbbbb`, i.e. (11 blanks.)

4. If the numeric value includes a fraction, no zero suppression will occur for zero suppression symbols specified to the right of a decimal point control symbol (".")
   e.g. For value 1.056, `FMT='zz.zzzzz'` returns character display `b1.05600`. (b = blank)

5. If the numeric value includes a fraction but has a zero integer value, and no digit control symbol ("9") exists to the left of the decimal point control symbol ("."), then a zero (0) will automatically be inserted immediately to the left of the decimal point in the character display.
   e.g. For value 0.056, `FMT='zNNN.zzzzz'` returns character display `bbb0.05600`.

Examples of zero suppression control symbol use:

Source Value (as decimal)	FORMAT String	Character Display Format
12	ZZZZZ	bbb12
37.8	ZZZ.Z	b37.8
37.8	ZZZ	b39
6.9	ZZZ.ZZZ	bb6.900
62	9ZZZZ	00062
0	ZZ.ZZ	bbbbb
0.0	ZZ.ZZ	bbbbb
0.021	ZZ.ZZZ	b0.021
0.021	.ZZZ	.021

Floating Sign Control Symbols

The floating sign control symbols "s", "S", "+" and "-" are FORMAT substitution control symbols used to represent the position of either a conditional decimal digit (0-9) or the numeric sign in the character display.

Floating sign control symbols operate in the same way as zero suppression symbols except that a numeric sign character is substituted in the position of the right most floating sign symbol for which zero suppression would occur.

If the floating sign symbol for which numeric sign substitution occurs is "s", "S" or "+", then the sign character substituted is "+" (plus) for positive numeric values and "-" (minus) for negative values. If this floating sign symbol is "-", then no substitution of numeric sign will occur for a positive value but character "-" (minus) is substituted if the value is negative.

Specification of a floating sign symbol as the left most substitution control symbol in the FORMAT string signals the intention to prefix the character display value with a sign character. Therefore, return code 8 is set and the character display format is set to asterisks (*) if decimal digit substitution has occurred for the left most floating sign symbol. i.e. substitution of a sign character cannot take place. The one exception to this is if the source numeric value is positive and the left most substitution control symbol is "-". In this case, no sign character would be substituted and so the character display is unaffected.

Special Case:
Floating sign control symbol "-" is treated as being a character constant if the character in the FORMAT string immediately to its left is a digit control symbol ("9"). This allows specification of "-" minus as a punctuation character or negative value suffix indicator in the character display.

Examples of floating sign control symbol use:

Source Value (as decimal)	FORMAT String	Character Display Format
32.8	SS9.99	+32.80
32.8	SS9.99	+32.80
85.6	SSSZZZZ.ZZ	bb+bb85.60
-0.07	+++ZZZZ9.99	bb-bbbb0.07
319.45	----9.99	bb319.45
222.67	--9.99	222.67
222.67	++9.99	******
213	ZZZSS9	bbb213
213	SSSSS9-9	bbb+21-3

Constants

Any character, within the format string, that is not one of the FORMAT **substitution control symbols** or the **decimal point control symbol** is treated as a character constant. A character constant represents the position of a conditional character of the same code point in the character display.

Character constants provide a method of embedding punctuation in the character display.

The interpretation of a character constant is determined by its location in relation to FORMAT substitution control symbols and the decimal point control symbol.

- If positioned to the left of the first occurrence of one of these control symbols, the character constant is substituted with a blank.
  e.g. For value 32.34, `FMT='ABCZZ9.99'` returns character display `bbbb32.34`.

  The only exception to this occurs for source numeric values where the integer part has a value of zero (0). In this case, a character constant within a FORMAT string which is located immediately to the left of the decimal point symbol (".") with no preceding digit control symbols ("9") is substituted with character zero (0).
  e.g. For value 0.56, `FMT='ABC.99'` returns character display `bb0.56`.

- If positioned to the right of FORMAT substitution control symbols, all of which have undergone zero suppression, the character constant is treated as being a zero suppression control symbol or a floating sign control symbol, if floating sign substitution is active.
  e.g. For value 923, `FMT='ss,ss9'` returns character display `bb+923`.

- If positioned between the first and last occurrences of a FORMAT substitution control symbol for which no zero suppression has occurred, the character constant is passed without translation into the same position of the character display.
  e.g. For value 432.34, `FMT='zABCz9.99'` returns character display `4ABC32.34`.

- If positioned following the last occurrence of a FORMAT substitution control symbol, the character constant is substituted with a blank if the source numeric value is positive and no substitution of numeric sign character "+" is to take place.
  e.g. For value 25, `FMT='zz9DB'` returns character display `b25bb`.

  Otherwise, the character constant is passed without translation into the same position of the character display.
  e.g. For value -25, `FMT='zz9DB'` returns character display `b25DB`.

Beware when using character constant "X" or "x" in a numeric FORMAT string. "X" and "x" are format control symbols for printable hex format strings. If the total number of "X" and "x" character constants is greater than the number of FORMAT substitution symbols, then a printable hex format string is assumed.

Examples of character constant use:

Source Value (as decimal)	FORMAT String	Character Display Format
54321	ZZ,ZZ9.99bCR	54,321.00bbb
-67381	ZZ,ZZ9.99bCR	67,381.00bCR
-76942.779	S,SSS,SS9.99bVAT	bb-76,942.78bVAT
-7.2	S,SSS,ZZ9.99bDEB	bbbbb-bb7.20bDEB
12	99.	12b
-12	99.	12.

Printable Hex FORMAT Symbols

Printable hex FORMAT strings may be used to format the character display of a source field as hexadecimal. i.e. 1 byte of the source field is represented by 2 hex digits in the character display.

The source field may be a character constant, a field definition or the source field containing the data typed value of a declared variable.

A printable hex character display format is applicable only for values that are the target of a CVCH operation or an element of an output (PRINT, PLOG, WRITE) operation. e.g.

```
DECLARE   BINVAR  BIN  INI=6589      * Binary integer (source field length=4)

PRINT "BINVAR Source = X'"    BINVAR FORMAT='xxxx,xxxx'   "'"
                              * Prints:  BINVAR Source = X'0000,19BD'

CVCH  'ABCD' ASCII    TO  1  FORMAT=" 'xx,xx xx,xx' = ASCII code. "
CVCH  'ABCD' EBCDIC   TO  1  FORMAT=" 'xx,xx xx,xx' = EBCDIC code. "
```

Printable hex FORMAT strings contain characters that represent either a **hex digit control symbol** or a **constant value** and may be applied to a source field of any data type.

The characters that comprise a printable hex FORMAT string are repeated as many times as is necessary to format all bytes of the source field. If FORMAT string repetition occurs, then the character display output terminates once all the source field bytes have been formatted. i.e. Any unprocessed, trailing characters belonging to a repeated FORMAT string are ignored. e.g.

```
PRINT  'AB' EBCDIC FORMAT='xx-->'    * Prints:  'C1-->C2'
```

While character constants and declared variable source fields each have an implicit length, a source field specified as a field definition must include an explicit length (or end position) value. This is because the length of a source field definition cannot be determined by the target field FORMAT string.

Hex Digit Control Symbol

The hex digit control symbol, "x" or "X" is used to represent the position of a hex digit (0-F) in the character display. Each hex digit is one of a pair representing a byte in the source field.

If "n" is the number of bytes in the source field and "m" is the number of hex digit symbols, then the FORMAT string will repeat if "m<2n". If "m>2n", then the additional trailing hex digit control symbols are upper cased and treated as character constants (i.e. character "X").

Examples of hex digit control symbols:

Source Field (as hex constant)	FORMAT String	Character Display Format
x'015A'	X	015A
x'002C'	XXX	002C
x'3132'	xxxxx	3132X

Constants

Any character, within the format string, that is not a hex digit control symbol is treated as a character constant. A character constant represents the position of a character of the same code point in the character display.

A character constant is passed without translation into the same position of the character display. This provides a method of embedding punctuation in the character display.

Examples of character constant use:

Source Field (as hex constant)	FORMAT String	Character Display Format
x'F1F2'	XX,	F1,F2
x'FF5A'	XXXX HEX	FF5A HEX
x'31323334'	XXXX,	3132,3334
x'41534349'	XXXXbASCIIb	4153bASCIIb4349
x'0001F301'	XX:XX**	00:01**F3:01
x'0001234C'	X.XX	0.001.234.C

DATE FORMAT Symbols

DATE FORMAT strings may be used to format the character display of a source date field. The source field may be a character constant or a field definition.

A DATE character display format is applicable only for character values that are the target of a CVDATE operation.

```
CVDATE     NOW                    TO   1  FORMAT="Day d(Ddd), ddth Mmm yyyy"
CVDATE   4 AT 21 TYPE=U STYLE=J  TO  81  FORMAT="yyyy/mm/dd"
```

DATE FORMAT strings contain characters that represent FORMAT **substitution control sequences** or **constant values**. This type of FORMAT string may apply to a source date field of any of the supported data types and styles.

Date Control Sequences

Date control sequences are used to represent the position of a date elements in the character display. Date element values are interpreted from the contents of the source date field.

Note that all date sequences are **case-sensitive** and are processed from left to right. The following table details all supported date substitution control sequences with examples of character display output based on a date of Friday 21st August 2015.

Date Control Sequence	Examples	Description
Mmmmmmmmm	Augustbbb	Month of year - '*Month*', where *"Month"* is the month name in mixed case, padded on the right with blanks up to a fixed length of 9 characters.
Mmmzzzzzz	August	Month of year - '*Month*', where *"Month"* is the month name in mixed case. The month name occupies an area of variable length up to a maximum of 9 characters. Trailing blanks are truncated so that the remainder of the character display output is shifted to the left as appropriate.
Ddddddddd	Fridaybbb	Day of week - '*Weekday*', where *"Weekday"* is the day name in mixed case, padded on the right with blanks up to a fixed length of 9 characters.
Dddzzzzzz	Friday	Day of week - '*Weekday*', where *"Weekday"* is the day name in mixed case. The day name occupies an area of variable length up to a maximum of 9 characters. Trailing blanks are truncated so that the remainder of the character display output is shifted to the left as appropriate.
yyyy	2015	4-digit year number.
ddth	04th	Day of month number in abbreviated sequence notation. i.e. *nn*st, *nn*nd, *nn*rd or *nn*th where *nn* is the day of month number. This substitution sequence occupies a fixed length area of 4 characters in the character display. For day numbers 1 through 9, the value is right adjusted with a leading blank.
zdth	1st	Day of month number in abbreviated sequence notation. Same as ddth except that the output length is variable. Day numbers 1 through 9 will occupy 3 characters with the remainder of the character display output shifted to the left as appropriate.

Mmm	Aug	Month of year - '*Month*', where *"Month"* is the month name in mixed case, truncated to a fixed length of 3 characters.
MMM	AUG	Month of year - '*Month*', where *"Month"* is the month name in upper case, truncated to a fixed length of 3 characters.
Ddd	Fri	Day of week - '*Weekday*', where *"Weekday"* is the day name in mixed case, truncated to a fixed length of 3 characters.
DDD	FRI	Day of week - '*Weekday*', where *"Weekday"* is the day name in upper case, truncated to a fixed length of 3 characters.
ddd	233	3-digit day of year number (001-366).
ww	33	2-digit week of year number (00-52). Sunday is day 1 of the week. If 1st January falls on a Thursday, Friday or Saturday (day of week numbers 5, 6 or 7), the first week of the year is week 0, otherwise it is week 1.
cc	20	2-digit century number.
yy	15	2-digit year of century number.
mm	08	2-digit month of year number (01-12).
dd	21	2-digit day of month number (01-31).
D	6	1-digit day of week number (1-7) where 1=Sunday, 2=Monday, ... 7=Saturday. Note that a "D" FORMAT control symbol is treated as a character constant if it occurs adjacent to an alpha character.
d	5	ISO 8601 standard 1-digit day of week number (1-7) where 1=Monday, 2=Tuesday, ... 7=Sunday. Note that a "d" FORMAT control symbol is treated as a character constant if it occurs adjacent to an alpha character.

Constants

Any character, within the format string, that is not part of a date control sequence is treated as a character constant. A character constant represents the position of a character of the same code point in the character display.

A character constant is passed without translation into the same position of the character display. This provides a method of embedding punctuation in the character display.

Examples of character constant use:

Source Field (as date constant)	FORMAT String	Character Display Format
2010/01/18	US Date: mm/dd/yyyy	US Date: 01/18/2010
1998/11/24	Julian Date: yyyy/ddd	Julian Date: 1998/328
1918/11/11	Day d (Ddd) ddth Mmm ccyy	Day 1 (Mon) 11th Nov 1918
2003/02/02	Dddzzzzzz the zdth of Mmmzzzzzz, ccyy	Sunday the 2nd of February, 2003

Expressions

An expression is a representation of a value or, in the case of regular expressions, a pattern of character data.

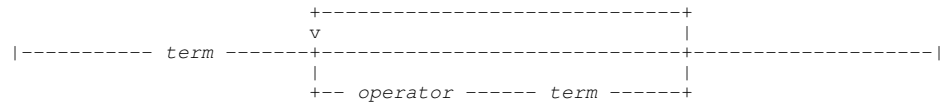
SELCOPY supports two types of expressions: Arithmetic Expressions and Regular Expressions

Arithmetic Expressions

An arithmetic expression represents a numeric value and can be one of the following:

- A single term specified by a constant or variable.
- A number of constant or variable terms and infix arithmetic operators.

Syntax:



Parameters:

term
: A constant or variable (declared, internal or @variable) of numeric or numeric character data type. Similarly, a substitution variable that gets substituted with a constant value or variable name of numeric or character numeric data type may also be used.

  Each term of the expression is an operand in an ADD or SUBTRACT function as defined by the infix *operator*. e.g. @x+1-5

operator
: One of the supported arithmetic operators, "+" (plus) or "-" (minus).

SELCOPY's arithmetic expressions are evaluated from left to right so that the result from an add/subtract function provides the first operand of the next add/subtract function. e.g. `A-B+C => Add(Subtract(A,B),C)`

Expression operands may be of different data types. However, the ADD and SUBTRACT functions can only operate on operands of the same data type and precision. If necessary, SELCOPY will convert expression operands to intermediate result values of the same data type and precision.

If any of the operands are of fixed or floating point data types, then all operands are represented internally by binary floating point values. If all operands are of integer data types then all operands are represented internally as integer values. The precisions of these internal floating point or integer values will be that of the expression operand having the greatest precision.

The result value is then converted, if necessary, to the data type of the target value. If necessary the result value will be rounded to the nearest fractional digit, as defined by the scale of the target. For integer targets, such as field positions and lengths, the result value is rounded to the nearest integer number.

Because variable values may change throughout the course of a SELCOPY program execution, any expression that involves variable terms is evaluated for each execution of an assignment or operation in which it has been specified.

The following contains examples of arithmetic expression use:

```
DECLARE    XVAR    BIN       INI=23           * Binary integer.
DECLARE    FMAX    FLT  BIN  INI=24.387       * Binary floating point.
DECLARE    FCUR    FLT  BIN                   * Binary floating point.
DECLARE    TEXT    CHA(25)                    * Character.

EQU   IREC     1

READ  INDATA   INTO IREC                      * Input file

PRINT        'Initial maximum allowed value: '  FMAX    STOPAFT=1

IF  POS IREC,   IREC+LRECL-1  =  'Value="'  PTR=@VAL           * Range test.
AND POS @VAL+6, @VAL+XVAR-1   =  '"'        PTR=@END  REVERSE  * Trailing quote.

  THEN  @LEN =  @END-1-@VAL-7+1               * Length of quoted text.
  THEN  TEXT =  POS @VAL+7   LENGTH=@LEN      * Assign unquoted text to TEXT.
  THEN PRINT 'Current value is: ' TEXT        * Print current value.
  ELSE GOTO GET                               * Get next input record.

CVCF    TEXT    TO   FCUR                     * Current value as floating point.

IF  FCUR    > FMAX+5.5                        * Compare with max value + 5.5.
  THEN FMAX = FCUR                            * Assign new maximum value.
  THEN SPACE
  THEN PRINT 'New     maximum allowed value: '  FMAX
```

Usage Note:

Provided no variable terms are used, an arithmetic expression may be specified as the value of an operation parameter resolved during control statement analysis (e.g. STEP, TIMES, STOPAFT). Therefore, the following may be used to limit the total number of times an operation is executed to 10, but allow for an additional number of executions passed as a parameter to the program.

```
OPTION  ENVFAIL='0'                       * Default of %1 is 0.
PRINT FROM   '---###---'  STOPAFT 10+%1   * Variable execution limit.
```

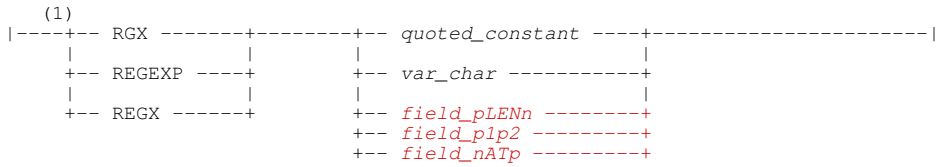
Regular Expressions

See also Appendix A. Regular Expression Summary for a quick reference of regular expression operators, text specifiers and predefined expressions.

A regular expression represents a complex pattern matching string for searching character data. First developed in the Unix environment, regular expressions may be used to search for string patterns in text. i.e.

- As a search string value on a character compare (IF, AND, OR) operation.
- As a search string value on a CHANGE operation.

If a regular expression is specified as the search string of a CHANGE operation, the replacement value may contain tag references to tagged sub-expressions of the regular expression search string pattern.



Note:

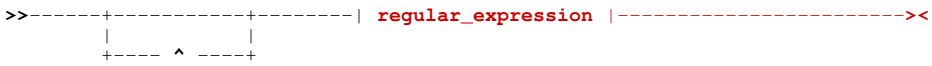
1. The RGX/REGEXP/REGX parameter keyword may occur anywhere following the operation keyword, provided it does not separate any other parameter from its argument.

A regular expression pattern string may be specified as a quoted character constant, a type 1/2/3 field definition or a declared variable of character data type.

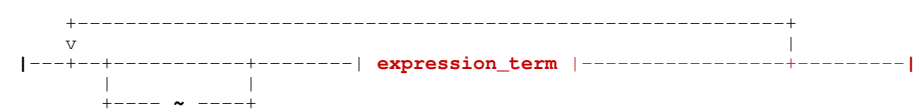
Note that, if specified as a variable or field definition, the expression must be evaluated for each execution of an operation on which it has been specified resulting in a significant overhead. Therefore, it is recommended that, whenever possible, regular expressions are specified as quoted character constants.

Regular Expressions Pattern String

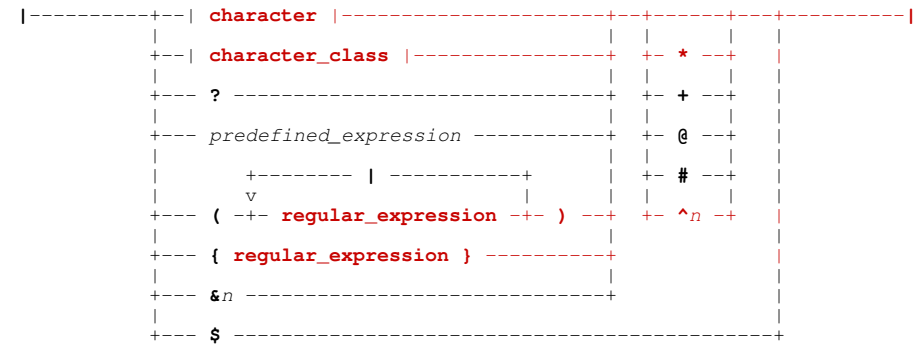
Syntax:



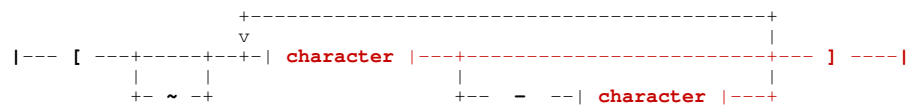
regular_expression:



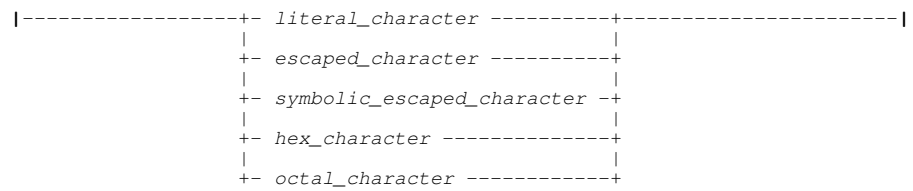
expression_term:



character_class:



character:



Note:

1. A blank character is a *literal_character* and should only be included in a regular expression pattern string if it is part of the pattern.

Parameters:

^

The "^" (**caret/circumflex**) character at the start of the pattern string indicates that a match for the pattern search string is successful only if it is found in the first position of the text being searched. e.g.

```
DECLARE  IREC     NTS(80)          * Null terminated character string.
IF       IREC   =  RGX '^[BKX]'    * True if 1st character is "B","K" or "X".
```

regular_expression

The expression or sub-expression comprising one or more character pattern matching expression terms. The terms of a *regular_expression* area evaluated from left to right.

~

The "~" (**tilde**) character represents the logical **NOT** operator and applies only to the *expression_term* that follows it.

If the character (or sequence of characters) in the source text does not match the pattern specified by *expression_term*, the test for that part of the regular expression pattern matching algorithm is TRUE. This type of test establishes only that the source text does not match the expression term and so does not influence the length of matched text. e.g.

```
CHANGE   IREC   RGX 'S~[CH]' '#'  * Change all occurrences of character "S"
                                  * that is not followed by a "C" or "H".
```

An *expression_term* that immediately follows an *expression_term* for which a logical NOT operation has been applied, will test source text starting at the same character offset. i.e. the offset within the source character text is not advanced following an *expression_term* test preceded by "~".

See also "~" (tilde) logical NOT operator in a *character_class* specification which has a different definition.

expression_term

A term within the regular expression constituting an individual pattern matching test.

Each *expression_term* tests a single character or sequence of characters in the source text. Furthermore, unless prefixed by logical NOT (~), each *expression_term* tests the source text that immediately follows the text matched by the previous *expression_term*.

character

A single character to be matched in the source text. The character can be expressed in the following ways:

literal_character

A character which represents a code point within the ASCII or EBCDIC code page used to specify the regular expression. If supplied as a quoted constant, then, by default, a regular expression will use the code page in which the control statements were entered.

A *literal_character* can be any character other than the regular expression special characters.

escaped_character

An escaped character is the escape operator "\" (backslash) followed by one of the regular expression special characters. This is used to define one of these syntactically significant characters as a character value. e.g.

```
DECLARE  DATA      CHARZ             INI = "ABC[[DEF]]G"
CHANGE   DATA   '[\[\]]'   NULL   * GIVES: "ABCDEFG"
```

symbolic_escaped_character
Symbolic escaped characters are used to represent print formatting characters. A *symbolic_escaped_character* is the escape operator \ (backslash) followed by one of the following lower case letters:

Symbol	Character	Description	ASCII code	EBCDIC code
b	BS	Backspace	X'08'	X'16'
f	FF	FormFeed	X'0C'	X'0C'
r	CR	Carriage Return	X'0D'	X'0D'
n	LF	LineFeed	X'0A'	X'25'
t	HT	Horizontal Tab	X'09'	X'05'

hex_character
A hex character is the escape operator "\" (backslash) followed by "**x**" or "**X**" and two hexadecimal digits for example **x00**. This provides a way of specifying a character which cannot be input from the keyboard.

octal_character
An octal character is the escape operator "\" (backslash) followed by one, two or three octal digits for example **072** or **72** (equivalent to **x3A**).

character_class
A character class is a set of one or more characters in the range **x00**-**xff** which represents a single character to be matched in the source text. A match occurs when the source text character matches one of those in the class.

The class definition is a list of characters enclosed within delimiters "**[ ]**" (square brackets). A character list may be specified as a number of single characters and/or character ranges.

Character ranges are defined as a range start character followed by a "**-**" (hyphen) followed by a range end character. A range comprises all hexadecimal values that fall between the character code points of the range limits.

An exception occurs when SELCOPY executes in **EBCDIC** based systems where non-alpha code points exist within the range of EBCDIC alpha character code points. If the range limits are alpha characters of the same (upper or lower) case, these non-alpha code points are disregarded. e.g. **H-K** represents the character range **HIJK** comprising code points \xC8, \xC9, \xD1 and \xD2 only.

Character ranges can be specified in ascending or descending order. e.g. **0-9** and **9-0** both define the set **0123456789** of numeric digits.

Any character, denoting a single character or the limit of a range of characters, may be specified in one of the formats described for *character*.

~
The "~" (**tilde**) character must be specified immediately following the opening square bracket "**[**", and as for *regular_expression*, represents the logical **NOT** operator.

When used in a *character_class*, it is equivalent to specifying all characters in the range \x00-\xff excluding those identified in the character class. Therefore, unlike *regular_expression*, a match in the source text for a *character_class* containing "~" increases the length of the matched text by one.

An *expression_term* that immediately follows the *character_class* will test source text starting at the next character offset. i.e. the offset within the source character text is advanced by one following a test for a *character_class* whether or not it contains "~".

```
CHANGE   IREC   RGX 'S[~CH]' '#'  * Change all occurrences of "Sx" to "#"
                                  * where "x" is not "C" or "H".
```

See also "~" (tilde) logical NOT operator in *regular_expression* which has a different definition.

?
The "?" (**question mark**) character is the wild card character representing any character code point. It is equivalent to the character class **[\x00-\xff]**.

predefined_expression
Several commonly used regular expressions have been supported as predefined expressions and can be referenced using the predefined expression operator **:** (colon) followed by a single lower case letter.

Supported predefined expressions are:

Name	Expression	Description
:a	[a-zA-Z0-9]	Alphanumeric character
:b	([\t]#)	White space (a string of blanks and tabs).
:c	[a-zA-Z]	Alphabetic character
:d	[0-9]	Numeric digit
:q	(("[~"]@")|('[~']@'))	Quoted string (in single or double quotes).
:w	([a-zA-Z]#)	Word (a string of alphabetic characters).
:z	([0-9]#)	Integer (a string of numeric digits).

(*regular_expression* **<** **\|** *regular_expression* **...** **>** **)**

An *expression_terms* may itself be a *regular_expression* in which case the *regular_expression* is a sub-expression which must be enclosed within "**( )**" (parentheses).

Since a sub-expression may comprise any number of *expression_terms* supported by *regular_expression*, it may be used to group together a number of terms to which the logical NOT operator "**~**" or any of the pattern repetition operators ("*****", "**+**", "**@**", "**#**" or "**^**") may be applied. e.g.

```
DECLARE     WK2  CHA                          INI "ABABABABABABABABAB, ABABAB"
CHANGE RGX  WK2  "(AB)^2" "xxxx" TIMES 1 * Gives: "xxxxABABABABABABAB, ABABAB"
CHANGE RGX  WK2  "(AB)#"  "yyyy"         * Gives: "xxxxyyyy, yyyy             "
```

Within the parentheses, the "|" (logical OR) character may be specified between a number of *regular_expression* sub-expressions to indicate **alternation**. When alternation is encountered, each of the sub-expressions is tested in turn against the same location within the source text until one matches or all fail. e.g.

```
DECLARE     WK2  CHA                          INI "A fox, an owl and the cow."
CHANGE RGX  WK2 "(:bthe:b|:ban:b)"  " "  * Gives: "A fox, owl and cow.       "
```

{ *regular_expression* **}**

A *regular_expression* enclosed within "**{ }**" (braces) is a tagged sub-expression.

Tagged sub-expressions provide a method whereby a portion of source text that matches the sub-expression may be referenced later in the same regular expression as another *expression_term*. Also, the portion of matching source text may be referenced in the replacement value of a CHANGE operation.

Up to 9 tagged sub-expressions may be specified. As regular expression pattern matching proceeds from left to right, each tagged sub-expression generates a tag reference sequence number starting at 1. Furthermore, the portion of source text that matches the sub-expression is saved and assigned to the reference number. Any occurrence of the tag reference thereafter is a reference to its assigned value. See **&***n* for tag reference specification.

By default, SELCOPY defines an implicit, tagged sub-expression for the entire regular expression specification for tag reference **&0**.

A tagged expression may encompass one or more groups of parenthesised *expression_terms*. e.g. `{(XYZ)^2}  {(XYZ)(AB)}`

Tagged sub-expressions may be nested so that one tagged sub-expression exists within another. If this is the case, the tag reference numbering sequence corresponds to the order in which tagged sub-expression "**{**" (opening brace) characters are encountered. e.g. `{X{YZ}}` `{AB{X^2}CD}` `{AB{(XYZ)#}}`

A tagged sub-expression may encompass a number of alternate sub-expressions. The value assigned to the tag reference number will be the source text matched by one of the sub-expressions. e.g. `{(X^2|Y|Z)}`

A tagged sub-expression may be a single sub-expression of an alternation. If so, the tag reference will be assigned a value only if the tagged sub-expression is the alternate sub-expression that matches the source text. Otherwise, the tag reference will be assigned to NULL. e.g. `({X}|Y|Z)` `({X}|{Y}|{Z})`

Note, however, that a tagged sub-expression must be a complete *expression_term*. e.g. The following are illegal: `{X}^2` `{(X}X)^2` `{(X|Y|Z})`

&*n*

The tagged expression reference operator **&** (ampersand) is used to refer to the source text matched by a previously defined tagged sub-expression. The integer *n* identifies the tagged sub-expression number (1-9) to which the tag reference applies.

The integer *n* must be in the range 0-9. **&0** is the SELCOPY defined tag reference assigned a value comprising all of the matched characters in the source text. **&1** to **&9** are assigned portions of the source text that match their corresponding tagged sub-expression definitions.

A tag reference may be included in the replacement value of a CHANGE operation to include portions (or all) of the matched source text in the value that substitutes the matching search value text. e.g.

```
DECLARE  WK2    NTS (26)                          INI "AB1BC"
CHANGE   WK2 RGX '{[A-Z]}[0-9]&1'  '--&0--&1-'  * Gives: "A--B1B--B-C"
```

In the above example, the CHANGE command operates on WK2 and the regular expression identifies 3 characters. The first character must be uppercase alpha and is tagged, the second must be numeric and the third character must be the same as the first.

$

The "$" (**dollar**) character may not be followed by a pattern repetition operator and indicates that the *expression_terms* that follow must match all remaining characters in the source text. e.g.

```
IF   IREC = RGX '$[BX]'    * True if the last character of IREC is "B" or "X".
```

Pattern Repetition Operators

The following operators are used to repeat the pattern matching term (character or sub-expression) that precedes it in the *expression_term* definition.

*****

The "*" (**asterisk**) character is the **minimal closure** repetition operator. It applies to the pattern matching term preceding it and specifies that **zero or more** occurrences of the pattern be included in the matched source text.

Only the **minimum** number of matched occurrences will be included in the final length of source text matched by the regular expression. If the last *expression_term* of the regular expression includes the minimal closure repetition operator, then the matched source text length for the *expression_term* will always be 0 (zero). e.g.

```
DECLARE  WK3    NTS (26)                    INI "X-ABBBBA-ABBBC-CD-E"
CHANGE   WK3 RGX 'AB*'  'x'  TIMES=1   * Gives: "X-xBBBBA-ABBBC-CD-E"
CHANGE   WK3 RGX 'AB*C' 'y'  TIMES=1   * Gives: "X-xBBBBA-y-CD-E"
CHANGE   WK3 RGX 'CB*D' 'z'  TIMES=1   * Gives: "X-xBBBBA-y-z-E"
```

@

The "@" (**commercial at**) character is the **maximal closure** repetition operator. It applies to the pattern matching term preceding it and specifies that **zero or more** occurrences of the pattern be included in the matched source text.

The **maximum** number of matched occurrences will be included in the final length of source text matched by the regular expression. e.g.

```
DECLARE  WK3    NTS (26)                    INI "X-ABBBBA-ABBBC-CD-E"
CHANGE   WK3 RGX 'AB@'  'x'  TIMES=1   * Gives: "X-xA-ABBBC-CD-E"
CHANGE   WK3 RGX 'AB@C' 'y'  TIMES=1   * Gives: "X-xA-y-CD-E"
CHANGE   WK3 RGX 'CB@D' 'z'  TIMES=1   * Gives: "X-xA-y-z-E"
```

+

The "+" (**plus**) character is the **minimal plus** repetition operator. It applies to the pattern matching term preceding it and specifies that **one or more** occurrences of the pattern be included in the matched source text.

Only the **minimum** number of matched occurrences will be included in the final length of source text matched by the regular expression. e.g.

```
DECLARE  WK3    NTS (26)                    INI "X-ABBBBA-ABBBC-CD-E"
CHANGE   WK3 RGX 'AB+'  'x'  TIMES=1   * Gives: "X-xBBBA-ABBBC-CD-E"
CHANGE   WK3 RGX 'AB+C' 'y'  TIMES=1   * Gives: "X-xBBBA-y-CD-E"
CHANGE   WK3 RGX 'CB+D' 'z'  TIMES=1   * Gives: "X-xBBBA-y-CD-E" (no match).
```

#

The "#" (**hash**) character is the **maximal plus** repetition operator. It applies to the pattern matching term preceding it and specifies that **one or more** occurrences of the pattern be included in the matched source text.

The **maximum** number of matched occurrences will be included in the final length of source text matched by the regular expression. e.g.

```
DECLARE  WK3    NTS (26)                    INI "X-ABBBBA-ABBBC-CD-E"
CHANGE   WK3 RGX 'AB#'  'x'  TIMES=1   * Gives: "X-xA-ABBBC-CD-E"
CHANGE   WK3 RGX 'AB#C' 'y'  TIMES=1   * Gives: "X-xA-y-CD-E"
CHANGE   WK3 RGX 'CB#D' 'z'  TIMES=1   * Gives: "X-xA-y-CD-E" (no match).
```

^*n*

A "^" (**caret/circumflex**) character that does not occur at the start of the pattern string is the **power** repetition operator. It must immediately be followed by a decimal integer value repetition factor, *n*, in the range 1-999. Note that the repetition factor may include leading zeroes and is terminated either by a non-numeric character or the end of the regular expression.

It applies to the pattern matching term preceding it and specifies that **exactly *n*** occurrences of the pattern be included in the matched source text. e.g.

```
DECLARE  IREC     NTS(80)         * Null terminated character string.
IF       IREC   =  RGX '[BKX]^3'  * "B","K" or "X" in 3 consecutive positions.
```

Value Assignment

Values are assigned to variables and field definitions that are nominated as the target of any input, arithmetic, data modification or conversion operation.

In particular, the MOD operation is used to initialise or update the value assigned to a target declared variable or field definition, using a source value specified as one of the following:

- A constant.
- A variable.
- A field definition.
- An arithmetic expression.

Where the source is specified as a variable or field definition, the value currently assigned to these objects will be assigned to the target object. e.g.

```
DECLARE  MAX_COUNT  BIN
MOD      MAX_COUNT      = 256          * Decimal integer constant value.
MOD      4 AT 1  TYPE=B = MAX_COUNT    * Value assigned to MAX_COUNT.
```

If the source object is a single, unquoted identifier that is not recognised as the name of a variable, then it is treated as being an unquoted literal or numeric constant, as appropriate to the data type of the target. e.g.

```
DECLARE  DEPT  CHA(3)
MOD      DEPT           = E12          * Unquoted literal constant value.
```

Unless the target of the MOD operation is a type 3 field definition, the MOD operator keyword may be omitted.

```
EQU      IREC    1

DECLARE  NAME   CHA(32)
DECLARE  OFFSET DEC(8)

NAME               = 'James Dean'       * Quoted character constant.
OFFSET             =  0x40              * Hex integer constant.
POS 23             = 'Marlon Brando'    * Character constant. (Length=13)
POS 51, 58 TYPE=P =  OFFSET-16          * Type 2 field definition.
OFFSET             =  8 AT 1 TYPE=F     * Floating point source field.

4 AT 101   TYPE=B =  233                * Type 3 field definition - Invalid.
```

Note: The initial value assigned to a declared variable may also be set using the INI parameter keyword of the DECLARE operation.

When assigning a value to an internal variable or @variable, specification of the MOD operator keyword is invalid and so **must** be omitted.

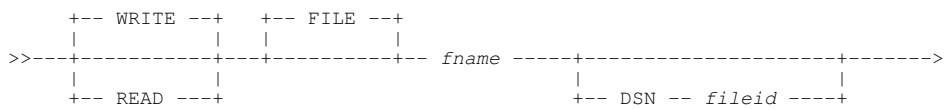
```
LRECL              =  256               * Internal variable LRECL.
RETCODE            =  33                * Internal variable RETCODE.
@VALUE             =  15                * @variable.
@HIT               =  IREC+LRECL        * Expression with internal variable.
```

If the target is a declared variable of numeric or numeric character data type, then rounding of the assigned value will occur if necessary. e.g. If assigning a rational numeric value to a variable of integer data type.

```
DECLARE  FLVAL  FLT(8)    BIN   INI='245.937646'
DECLARE  DECVAL DEC(6,3)
DECLARE  BINVAL BIN(4)

DECVAL             =  FLTVAL            * Value: 245.938
BINVAL             =  DECVAL            * Value: 246
```

Input/Output Data Objects

Primarily, SELCOPY is used to process data read from one or more data source objects and then write data to other data objects. Detailed syntax for input and output may be found in dcoumentation for the READ and WRITE operations. However, for file (data set) I/O the important parameter elements are as follow:



The types of data objects supported by SELCOPY for input/output and their specification within controls statements are documented below.

Files

Both input and output are supported for data stored as files on mounted direct access storage devices (e.g. hard disk, USB, CD, DVD) and belonging to any file system supported by the environment architecture. Furthermore, for systems that support device independence (e.g. z/OS, Linux, Unix), I/O is supported for files on mounted tape devices.

File Id

A file Id is a name that identifies a file and its location within a file system. If specified on a SELCOPY control statement, a file id is a programmer defined name identifier.

For the z/OS **MVS file system**, files are cataloged **data sets** and are each identified by an up to 44 character data set name (DSN) comprising a number of up to 8 character qualifiers separated by dots, e.g. `CBL.SELCOPY.TEST03.KSDS`. If the data set is partitioned (i.e. a PDS or PDSE library), then the file may include a trailing, up to 8 character member name reference in parentheses. e.g. `CBL.SELCOPY.SYSIN(SSTEST05)`

For the **VM/CMS file system**, files on a CMS mini-disk are identified by a file name, file type and file mode ('*fn ft fm*') delimited by one or more intervening blanks. e.g. `CBLINS EXEC A1`

For **Unix-like file systems**, including the CMS byte file system (BFS) and z/OS hierarchical file systems (HFS/ZFS), files are identified by a case sensitive, hierarchical file path which includes an implicit or explicit directory path. e.g. `/home/nbj/sample_text_v32`

For **Microsoft Windows file systems**, files are identified by a case insensitive, hierarchical file path which includes an implicit or explicit drive letter and directory (folder) path. e.g. `c:\tmp\htmlpdf\zzslprog.html`

Throughout this document, the term *file* applies to CMS, Windows, Unix and Linux files and MVS data sets. Similarly, the term *fileid* applies to the name by which a file on these file systems are identified.

Fileid Clause

File ids referencing a local or networked file, may be specified as arguments to SELCOPY input parameters and on a number of SELCOPY options and operations.

A *fileid* may be specified as an unquoted literal, quoted character constant or, depending on the operation classification, a field definition or declared variable of character data type.

In contrast to SELCOPY's usual processing, a *fileid* specified as an unquoted literal will not be upper cased and may contain asterisk (*) characters without it denoting the start of comment text. However, unquoted *fileids* may not contain blanks or parentheses and may be interpreted differently to quoted *fileids*, as described for MVS file system data sets below.

A *fileid* specified as a field definition must be of field Type 1, Type 2 or Type 3. Use of a field definition or variable to identify a *fileid* has the advantage of being dynamic. Therefore, the *fileid* assigned to a file name can change throughout the course of the program execution. Because the *fileid* value is unknown at control statement analysis, the file open is deferred until selection time processing. Furthermore, any change in the *fileid* value will take effect only when a specific open is executed for the file name.

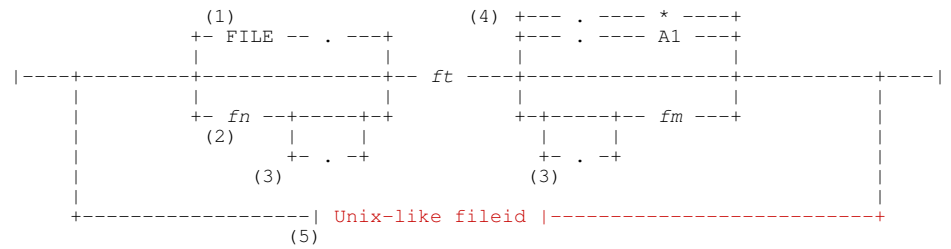
For READ DIR and DIRDATA input, *fileid* may contain wildcard characters to represent a fileid mask. Only fileids that match this mask will be processed. Fileid masks may have the following characteristics:

- CMS mini-disk *fileids* support asterisks (*), representing zero or more characters, in the *fn* and *ft* qualifiers and as the *fm* qualifier. They also support percent (%), representing exactly 1 character in the *fn* and *ft* qualifiers.
- z/OS MVS file system *fileids* support asterisks (*), which represent zero or more characters in a library member name.

- Unix-like file system and Windows file system *fileids* support asterisks (*), representing zero or more characters, in the file name but not in the directory names. Question mark (?) is also supported, representing exactly 1 character in the file name.
- Windows file system *fileids* support specification of multiple disk letters.

Syntax of fileid on each operating system supported by SELCOPY follows.

VM/CMS fileid:



VM/CMS fileid Syntax Notes:

(1) **CMS**: An *fn* of "FILE" is default only if:

1. *fileid* comprises an *ft* only. (i.e. no *fm* is specified.)
2. *fileid* is an unquoted literal or a quoted character constant.
3. If specified as a quoted character constant, *fileid* contains no leading or trailing blanks.
4. *fileid* (*ft*) is not a defined FILEDEF or DLBL ddname.

If *fn* is not specified and no default is assigned, then *ft* is treated as being the *fn* value, *fm* becomes the *ft* value and *ft* defaults as described in syntax note (4) below.

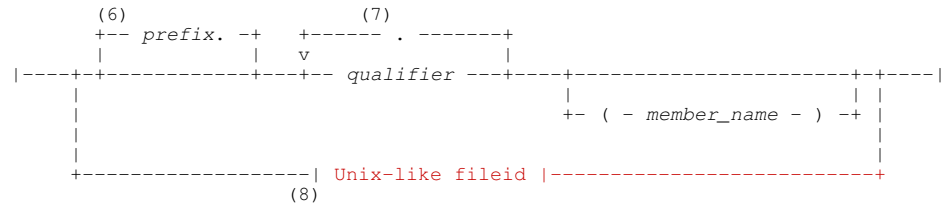
(2) **CMS**: *fn* is mandatory when referencing a CMS mini-disk *fileid* using a field definition.

(3) **CMS**: A single dot (.) with no intervening blanks may be used to delimit the *fn*, *ft* and *fm* of a CMS mini-disk *fileid*. If *fileid* is an unquoted literal, then use of this dotted notation is mandatory.

(4) **CMS**: If omitted, *fm* will default to asterisk (*) on input, and to A1 on output.

(5) **CMS**: If *fileid* does not match the format criteria of a CMS mini-disk *fileid* or is not found on an accessed disk, any default values for *fn* and *fm* are dropped from the *fileid* and it is treated as a Unix-like file path. This identifies a file on the CMS byte file system (BFS) or networked file system (NFS).

z/OS fileid:



z/OS fileid Syntax Notes:

(6) **z/OS**: A *prefix* qualifier equal to the defined TSO prefix, otherwise the ACF userid associated with the SELCOPY execution, is automatically included in *fileid* if:

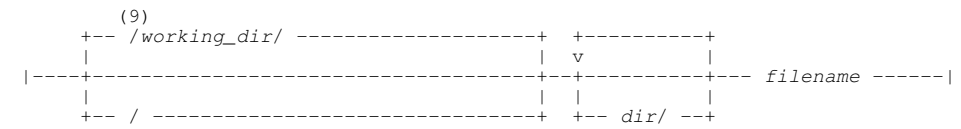
1. *fileid* is an unquoted literal or a field definition which does not include enclosing quotes or apostrophes as part of the field text.
2. Environment option DSNPFX=YES (the default) is in effect.

If a *prefix* is included but the open of *fileid* fails, then the open is reattempted for *fileid* without *prefix*. This subsequent attempt to open *fileid* may be suppressed by explicitly specifying DSNPFX=YES on the I/O operation.

(7) **z/OS**: A single dot (.) is used with no intervening blanks to separate each, up to 8 character qualifier in the up to 44 character MVS data set name *fileid*.

(8) **z/OS**: If *fileid* does not match the format criteria of an MVS data set name (with an optional library member name) or is not recognised as a cataloged data set, any default prefix value is dropped from the *fileid* and it is treated as a Unix-like file path. This then identifies a file in the z/OS hierarchical file system (HFS or ZFS) or a mounted networked file system (NFS).

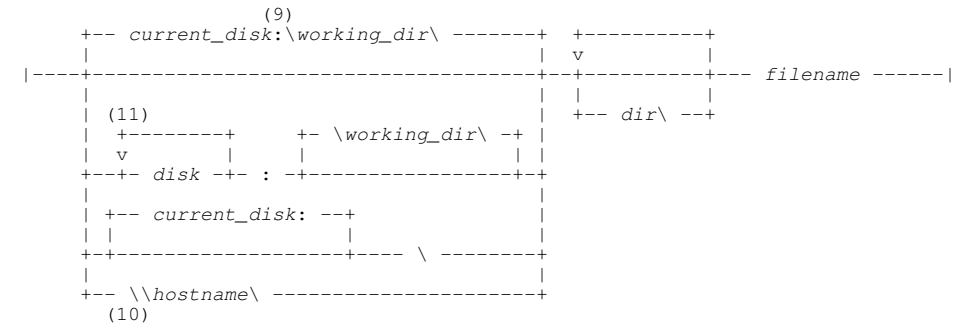
Unix-like fileid:



Unix-like fileid Syntax Notes:

(9) A single slash (/) or backslash (\) is used with no intervening blanks to separate each level of directory in the file path. One or other of these characters, or a combination of both, may be used in any Unix or Windows *fileid* specification.

Windows fileid:



Windows fileid Syntax Notes:

(10) Double slash (//) or backslash (\\) at the start of the *fileid* identifies it as a UNC (Universal Naming Convention) file path which starts with a networked computer host name specification.

(11) Multiple disk letters may be specified only on an input *fileid* being read with DIR or DIRDATA. This indentifies the *fileid* as a mask used to select matching *fileid*s from across multiple disks.

Parameters:

`VM/CMS fileid`

fn
: An up to 8 character CMS mini-disk file name.

ft
: An up to 8 character CMS mini-disk file type.

fn and *ft* may be separated a single "." (dot/period) with no intervening blanks or, unless *fileid* is specified as an unquoted literal, any number of blanks.

fm
: A 1 or 2 character CMS mini-disk file mode. The file mode identifies the disk letter on which a mini-disk volume is accessed.

ft and *fm* may be separated a single "." (dot/period) with no intervening blanks or, unless *fileid* is specified as an unquoted literal, any number of blanks.

On input only, asterisk (*) may be specified for *fm* so that SELCOPY scans each accessed CMS volume in alphabetical order for the specified *fn.ft*, and processes the first occurrence found.

`z/OS fileid`

prefix
: The *prefix* value identifies the high level qualifiers that SELCOPY automatically prefixes to an unquoted data set name *fileid* specified in the program or via a SELCOPY command parameter.

If executing in a z/OS TSO environment, *prefix* is the defined TSO prefix value. In all other environments (e.g. batch), *prefix* is the prevailing ACF userid.

qualifier
: An up to 8 character MVS data set name (DSN) qualifier. Each qualifier specified must be separated by a "." (dot/period) with no intervening blanks. Any number of qualifiers may be specified to identify an existing, cataloged data set.

The total length of all qualifiers, including separating dots and implied *prefix*, may not exceed 44 characters.

member_name
An up to 8 character MVS PDS or PDSE library member name enclosed in "( )" (parentheses).

The closing parenthesis ")" must be one of the 9 characters following the opening parenthesis "(" and *member_name* must immediately follow the opening parenthesis to satisfy SELCOPY's interpretation of *fileid* as a PDS/PDSE member reference.

Unless specified as an unquoted literal, any number of blanks may exist between the end of the library DSN and the opening parenthesis "(" and also between the end of *member_name* and the closing parenthesis ")".

`Unix-like fileid`
Identifies a *fileid* that obeys the POSIX standard. Note that the file directory names that comprise these types of *fileid* are case sensitive. For this reason, unquoted literal *fileids* are not upper cased by SELCOPY.

working_dir
The current (present) working directory as identified by the *pwd* Unix shell command.

The *working_dir* includes all directories in the working directory hierarchy starting from the root directory "/". It is included as the *fileid* prefix only if the first character of the *fileid* is not "/" (slash) or "\" (backslash).

Unless specifically changed, *working_dir* is usually the user's defined home directory.

dir/
A directory within the file path directory hierarchy. Each *dir* specification must be followed by a "/" slash or "\" backslash, separating it from the next directory specification or the file name.

If *fileid* begins with "/" or "\", the first *dir* specification corresponds to a directory in the root directory.

filename
The name of the file to be processed.

`Windows fileid`
current_disk
The current disk letter.

current_disk is included as the *fileid* prefix only if the first character of the *fileid* is not "//" (double slash), "\\" (double backslash) or a *disk* specification.

working_dir
The current directory as identified by the *chdir* Windows command shell command with no parameters.

The *working_dir* includes all directories in the current directory hierarchy starting from the specified *disk* or the *current_disk*. It is included only if the first character of *fileid* or the first character following a *disk* specification is not a "/" (slash) or a "\" (backslash).

disk:
A disk volume letter followed by ":" (colon).

A number of *disk* letters may be specified for READ DIR or DIRDATA input only.

hostname\
A networked resource server name prefixed by "//" (double slash) or "\\" (double backslash), and followed by a single "/" slash or "\" backslash.

Use of *hostname* identifies the *fileid* as a network file referenced using the network Universal Naming Convention (UNC).

dir/
A directory within the file path directory hierarchy. Each *dir* specification must be followed by a "/" slash or "\" backslash, separating it from the next directory specification or the file name.

If *fileid* begins with "/" or "\", the first *dir* specification corresponds to a directory in the root directory of *disk* or *current_disk*.

filename
The name of the file to be processed.

File Name

A file name is a mandatory, up to 8 character programmer defined name used by SELCOPY to reference an individual input or output data object. Throughout this document, the term *fname* is used to represent a file name.

An explicit or implicit *fname* specification is mandatory on all SELCOPY I/O operations. Also, since SELCOPY is capable of processing a number of input and output data objects simultaneously, *fname* may be specified on condition operations and sub-operations (IF, AND, OR) to test the current status of a specific input file. e.g. `IF EOF` and `IF INCOUNT`. If no *fname* is

specified on these types of operations, the prime input object is implied.

All data objects must have an assigned *fname*. For z/OS data sets and PDS members, this must be the name (ddname) that is allocated to the DSN. This name may be allocated prior to executing SELCOPY or may be dynamically allocated by SELCOPY using dynamic allocation parameters on I/O operation control statements. (See Dynamic Allocation.)

Similarly in z/VM CMS, in order to process a file on an accessed z/OS or VSE volume, *fname* must be a name (ddname) that has already been allocated to the file via the FILEDEF z/VM CMS command. It is not necessary to have pre-allocated *fname* for files belonging to the CMS format file system.

If a file already has an allocated ddname, SELCOPY I/O operations should reference this name as *fname* with no **DSN**=*fileid* specification. If no such allocation exists (as is always the case for Windows, Linux and Unix systems), at least one statement containing an I/O operation must exist that associates an *fname* with the file's *fileid*.

```
READ   INFILE   DSN='/home/nbj/Documents/readme_selcopy'
WRITE  PDSMBR   DSN='CBL.NBJ.JCL(XXJOB01)'              * Dynamic Allocation.
```

STDIN and STDOUT

STDIN and STDOUT refer to the standard input (stdin) and output (stdout) streams for the SELCOPY process. In z/OS and VM/CMS (non-POSIX) environments, STDIN and SYSIN are synonymous.

So long as SELCOPY's input control statements are not passed via stdin (see SELCOPY parameter **-ctl**), then standard input stream may be read by SELCOPY as data. Similarly, if SELCOPY list output is not directed to stdout (parameter **-lst**), then data may be written to the standard output stream.

This is of particular use in POSIX environments (e.g. Linux and Unix) and in Microsoft Windows where support exists for pipelines, a chaining of processes by their standard input (stdin) and output (stdout) streams.

SELCOPY may be one such process so that data piped to it from another process can be read using input operation, READ STDIN. Similarly, standard output may be piped to another process using output operation, WRITE STDOUT.

Stream input ends when the Unix <Ctrl+D>, end-of-transmission (EOT) control character; the Windows <Ctrl+Z>, end-of-file (EOF) control character; or end of SYSIN input is encountered. If stdin input is not passed by another process, then input is taken from the keyboard that started the SELCOPY process. Similarly, if stdout is not passed to another process, output is written to the terminal.

```
ls -l  |  selcopy \!OPT NOBAN \!READ STDIN \!IF POS 1,30 = "'nbj'" \!THEN WRITE STDOUT \!END   | sort -
```

Database Tables

Supported only for SELCOPY on Microsoft Windows, database and spreadsheet tables may be processed for input and output via an ODBC (Open Database Connectivity) client interface. To do this, SELCOPY relies on the appropriate ODBC drivers and client software having been successfully installed and configured on the local system.

Database servers (e.g. DB2, Oracle, MySQL, SQL Server) to which a connection will be made and from which data will processed, must also have been configured to support access via ODBC.

SELCOPY submits SQL (Structured Query Language) statements to the ODBC interface in order to perform I/O on result tables. It can also perform immediate execution of SQL statements supported by the database management service. e.g. Commit changes as well as create and drop database objects.

Like files, a database input result table (a set of columns and rows generated by an SQL query statement) and an output base table must have an associated *fname*. Furthermore, at least one I/O operation specifying either **TABLE**=*table_name* or **SQL**=*select_clause* must exist to identify a database table object processed by SELCOPY.

```
READ  INDB1  TABLE="user_objects"   WHERE="object_type = 'TABLE'"
READ  INDB2    SQL="SELECT *  FROM CBL.APIFUNC"

WRITE INDB3  TABLE="ZZS.ZZSIQ"   FROM 1001
```

Note that, regardless of a table column's source data type, all input table data is automatically converted to character representation. Likewise, data referenced as the source values of an output operation to a database table must also be in character format as specified by SQL INSERT VALUES syntax.

By default, a declared variable of fixed length character data type is automatically defined for each selected input table column.

Each of these variables has the same name as the column to which it applies and, by default, the variables' source fields occupy consecutive areas starting at position 1 of the work area or input buffer. If the table rows are read into any other position (READ INTO=*n*), reference to any of the input table column variable names must include an offset to the position of the input data. (i.e. *varname*+*n*-1)

SELCOPYi Lists

Supported only on z/OS and VM/CMS systems where SELCOPY (SLC) and SELCOPYi are both components of the SELCOPY Product Suite, generated list output, as produced by SELCOPYi, may be processed as input by the SLC program.

SLC loads and calls the SLCLIST module to interface with the SELCOPYi list functions Lists produced by SELCOPYi include SMS storage groups, DASD volumes, VTOC entries, cataloged data sets, library members, HFS files, allocated data sets, enqueued resources, APF authorised libraries and link list libraries. Each list may be filtered and sorted as required and may include a subset of the standard list columns.

Like files, SELCOPYi list input must have an associated *fname*. Furthermore, at least one SELCOPY READ operation specifying **LIST**=*list_command* must exist for each input list object processed by SELCOPY.

The standard list column headers and *list_command* primary command syntax, appropriate to each type of list, is documented in chapter *"SELCOPYi Command Reference"* of the *"SELCOPYi Reference and User Guide"*.

```
READ  INLIST1  LIST="LSG"              * List all SMS Storage Groups.

READ  INLIST2  LIST="LD CBL.SSC.**"        WHERE="RECFM='FB' AND LRECL=80"
READ  INLIST3  LIST="LL CBL.SSC.CTL(SQ*)"  WHERE="CURSIZE > 36"
```

All data read from the list is in character format and, by default, a declared variable of fixed length character data type is automatically defined for each selected list column. If the column data is numeric in nature, the variable is of numeric character data type.

Each of these variables has the same name as the column to which it applies and, by default, the variable source fields occupy consecutive areas starting at position 1 of the work area or input buffer. If the list records are read into any other position (READ INTO=*n*), reference to any of the list input variable names must include an offset to the position of the input data. (i.e. *varname*+*n*-1)

VSAM Files

VSAM files (data sets) are formats of file processed using IBM's Virtual Storage Access Method. VSAM data set organisations supported by SELCOPY are as follow:

- Key Sequenced Data Set (KSDS)
- Relative Record Data Set (RRDS)
- Entry Sequenced Data Set (ESDS)

VSAM data sets are native to z/OS systems and no configuration is required for SELCOPY access. SELCOPY will automatically detect whether a data set is VSAM managed and process it accordingly.

On VM/CMS, VSAM file I/O is supported via the VSE/VSAM program product which must be installed and accessible to the executing SELCOPY program for successful processing of OS and VSE VSAM files. Additionally, the *fileid* of any VSAM file processed by SELCOPY must first be allocated to *fname* via a DLBL CMS command with option VSAM. Note that VSE/VSAM is no longer marketed by IBM, however, continues to be supported on current versions of z/VM.

On Microsoft Windows, SELCOPY supports I/O on all the sequential, relative and indexed format files provided by Micro Focus® products. In particular, the relative and indexed file formats offer similar structure to the RRDS, ESDS and KSDS file formats provided by IBM VSAM. Micro Focus run-time libraries CBLRTSS.DLL and MFFH.DLL must be available in the search PATH during execution of SELCOPY in order to process these types of file. Furthermore, SELCOPY environment option MFC must be set and parameter keyword, VSAM, specified on at least one SELCOPY I/O operation that references the file.

```
OPTION  MFC
READ    INVS   DSN="z:\cc\mfc\TESTDATA"   VSAM
```

Windows Keyboard

Supported only for SELCOPY on Microsoft Windows, a SELCOPY program may output key strokes to the system as though typed from the local keyboard input device.

This feature provides a method by which text may be dynamically entered in a specific window that is open on the same host in which SELCOPY executes. This may be a command shell or an application window such as a web browser or 3270 emulator. If the required window is not open, SELCOPY's SYSTEM command may be used to start the window before writing key strokes to it.

Like files, a single output key stroke stream must have an associated *fname*. Furthermore, at least one I/O operation specifying either **WIN**=*window_title* must exist to identify the open window to which key strokes will be directed. Any execution of an output operation on this *fname* will always switch focus to this window before key strokes are passed.

```
WRITE ZOS      WIN='Session A'   'LOGON APPLID(SELCOPYI)[CR]'
WRITE ZOS                        'USER123[CR]'

WRITE SEARCH   WIN='Google'      FROM  50 AT 101
```

Windows Clipboard

Supported only for SELCOPY on Microsoft Windows, text in the Windows system clipboard may be processed for input and output using READ CLIP and WRITE CLIP respectively.

This may be of particular use where key strokes have been written to a different window resulting in the copying of selected data to the clipboard. The SELCOPY program can then read this data from the clipboard for subsequent processing.

READ CLIP is allocated *fname* CLIPr and WRITE CLIP is allocated *fname* CLIPw.

```
READ  CLIP      INTO 1

WRITE CLIP      FROM 1001
```

Data Record Format

Data in file objects is usually organised into a number of distinct lines referred to as data records. The term, record, is used throughout SELCOPY documentation to refer to input and output lines of data (file records, database table rows, LIST input rows, etc.)

The format in which record data exists within a file object on a storage device can vary, however, it must be of a format supported by a file access method provided by the system. On all operating systems, SELCOPY treats data objects as being of fixed length format (RECFM F), variable length format (RECFM V) or undefined format (RECFM U).

Data objects of fixed length format contain records that are all of the same defined fixed length value. Variable length format data objects contain records of potentially different, self-defining lengths up to a defined maximum (each record is prefixed by a field containing the length of the record data). Data objects of undefined format contain data from which record limits do not exist or are determined by standards established by individual operating systems, applications or access methods.

SELCOPY I/O operations support parameter RECFM to specify the record format of the data object being processed. However, RECFM should not be specified for the following types of data object where SELCOPY can automatically establish the record format required:

- A SELCOPYi list or ODBC database table object. These are always of fixed length as established by the sum of the column data maximum widths.
- A z/OS data set that is not managed by VSAM (e.g. a sequential data set or a PDS/PDSE library member.) For these types of data set, a RECFM value is usually assigned and recorded in the VTOC (Volume Table of Contents) when the data set is first allocated.
- An existing z/VM CMS file. For this type of file object, a RECFM value is usually assigned and recorded in the CMS FST (File Status Table) when the file is created.
- An IBM VSAM managed data set or Micro Focus file accessed via the Micro Focus file handler. These are of undefined format (RECFM U) where the actual format of the data records is determined by the access method.
- A Windows or Unix-like file that contains records which are delimited by end-of-line characters (the standard format for text in files of this type). By default, SELCOPY input treats these types of file as undefined format (RECFM U).

RECFM should be specified on a SELCOPY I/O operation where the required record format cannot automatically be established or is not the default used by SELCOPY. e.g.

- A new z/VM CMS file.
- A ddname representing a z/VM CMS OS simulated data set which is not managed by VSAM and for which no RECFM has been defined (via the CMS FILEDEF command).
- A Windows or Unix-like output file that contains end-of-line characters but the prime input object is not RECFM U.
- A Windows or Unix-like file that does not contain end-of-line characters. i.e. Records may be of fixed length (RECFM F) or be of one of the supported variable length formats (RECFM V, VB, V2, V3 or MFV).

File objects that are of fixed or variable length record format may also be structured so that consecutive records are grouped together in a block. File objects of this type are assigned a record format of RECFM FB or RECFM VB respectively. File objects supporting a blocked record format are as follow:

- z/OS data sets and z/VM CMS OS simulated data sets.
- Windows or Unix-like output files that are of variable length record format only.

The following sections include a detailed description of each record format, their relationship with BLKSIZE and LRECL parameters and the method by which record lengths are determined.

Fixed Length Record Format

Records are all of the same fixed length. This type of record organisation is applicable to the following file definitions:

- z/OS and z/VM CMS OS simulated data sets allocated as RECFM F or FB.
- z/OS and z/VM CMS VSAM data sets defined as a RRDS (Relative Record Data Set).
- z/VM CMS files defined as RECFM F
- Windows and Unix-like files processed by SELCOPY I/O with parameter RECFM F or FB.
- Micro Focus Sequential and Relative defined as having fixed length records.
- SELCOPYi List and ODBC table objects.

The length of each VSAM RRDS data record processed by a SELCOPY I/O operation is the defined RECORDSIZE value. For other file types, the record length is defined by the allocated LRECL value or the value specified by the LRECL parameter on the SELCOPY I/O operation itself.

For z/OS and z/VM CMS OS simulated non-VSAM data sets, a file containing blocked, fixed length records must be allocated with RECFM FB and a BLKSIZE value which is a multiple of the fixed record length (LRECL). This BLKSIZE value determines the size of the I/O buffer used by access method services.

Specific blocking of z/VM CMS RECFM F files is not supported. For these types of file, all internal data management such as blocking is handled by CMS.

For Windows, Unix-like and Micro Focus files, there is no concept of fixed length record blocking and so RECFM F and RECFM FB are synonymous. SELCOPY reads or writes fixed length data using an I/O buffer which has size determined by the BLKSIZE parameter (default 2048 bytes). The buffer size does not need to be a multiple of the LRECL value since SELCOPY I/O processing copes with the possibility of incomplete record data occupying the last bytes of the buffer.

When reading a Windows or Unix-like file with parameter RECFM F, the length of the last record in the file may be shorter than the specified input LRECL. If this is the case, then following the READ operation the value of the internal variable, LRECL, will be the length of the short record.

Output to any RECFM F file will truncate long records and pad short records to the fixed length using the FILL character. Windows or Unix end-of-line characters will not be appended to RECFM F output records. If record padding is not desirable, then RECFM U output should be used with EOL NO. e.g.

```
READ  ABC    LRECL=100       * Input file of 441 bytes, LRECL implies RECFM F.
WRITE OUTF   LRECL=100       * Writes 5 RECFM F records, all of length 100.
WRITE OUTU  RECFM U  EOL=NO  * Writes 4 records of length 100, 1 of length 41.
```

Variable Length Record Format

Records may be of different lengths. This is type of record organisation is applicable to the following file definitions:

- z/OS and z/VM CMS OS simulated data sets allocated as RECFM V or VB.
- z/OS and z/VM CMS VSAM data sets defined as a KSDS (Key Sequenced Data Set) or ESDS (Entry Sequenced Data Set).
- z/VM CMS files defined as RECFM V
- Windows and Unix-like files processed by SELCOPY I/O with parameter RECFM V, VB, V2, V3 or MFV.
- Micro Focus Record Sequential, Relative and Indexed files defined as having variable length records.

The lengths of records processed by VSAM I/O on a KSDS or ESDS may vary up to the data set's maximum defined RECORDSIZE value. Note that SELCOPY reports this maximum value in the BLKSIZE column of the list output summary block. Similarly, Micro Focus format I/O will return records with varying lengths as determined by the file's defined organisation.

z/OS and z/VM CMS OS simulated data sets of RECFM V or VB have a maximum record length as defined by the allocated LRECL value. This value includes the length of the 4-byte binary record length field prefix known as the RDW (Record Descriptor Word). When records are read from a RECFM V or VB data set, the SELCOPY option RDW/NORDW determines whether the RDW field is included in the record data. If the records are to be organised in blocks, the data set must be allocated with RECFM VB and a BLKSIZE value which defines the maximum size of the record blocks. Each block of records is prefixed by a 4-byte binary block length field known as a BDW (Block Descriptor Word). The allocated BLKSIZE value determines the size of the I/O buffer used by access method services.

Records belonging to z/VM CMS files defined as RECFM V do not have an RDW and have no specific BLKSIZE to define record blocks. For these types of file, all internal data management, including blocking, is handled by CMS. For this reason, the record format is undefined to SELCOPY and so RECFM V is synonymous with RECFM U. CMS RECFM V files are reported as being RECFM U in the list output summary block.

Variable length record processing, where each record has a length field prefix, is also supported by SELCOPY I/O for Windows and Unix-like files. Because record formatting of this type is not standard for these types of file, SELCOPY must perform additional processing to establish individual record lengths. A number of variable record formats are supported, and so the required RECFM type must be specified on the SELCOPY I/O operation as follows:

RECFM V

Processes variable length records that are exactly the same format as that supported by RECFM V data sets on the z/OS platform (i.e. records are prefixed by a 4-byte binary RDW.) This is of particular use in processing data set ported from a mainframe environment.

By default, specifying RECFM V on a SELCOPY I/O operation is equivalent to specifying RECFM VB, unless option NOBDW is also specified. NOBDW indicates that the records are unblocked.

RECFM VB

Equivalent to RECFM V with option BDW (the default setting) applied.

The specified BLKSIZE value (default 2048) is used to define the maximum size of a block and so the I/O buffer size for the file. Records are de-blocked (input) or blocked (output) by stripping or adding a BDW and RDWs as applicable.

On input, if a BDW contains a value greater than the BLKSIZE value, ERROR 537 (RECFM=V BDW/RDW value exceeds BLKSIZE or LRECL) is returned.

RECFM V2

Similar to RECFM V with option NOBDW, RECFM V2 processes unblocked, variable length records that have 2-byte binary length fields. Unlike RDW for RECFM V, the record length field values do not include their own (2-byte) length, but only the length of the data.

RECFM V2 was introduced to process files supported by certain COBOL compilers and SORT utilities on Windows and Unix platforms.

Example of RECFM V2 input:

```
READ  'rcfm-v2.fil'        INTO 5  RECFM V2  RDW * Override default NORDW.
WRITE 'c:\tmp\rcfm-u.tmp'  FROM 5  RECFM U       * Kills 2-byte RDW, uses EOL.
CVCH   2 AT 5              TO   1                * Get readable RDW.
POS         5    =  ' '                          * Separate prefix from data.
LRECL            =  LRECL+4       * Account for extra 4 bytes for hex prefix.
PRINT                             * Data Record with hex prefix giving length.
```

RECFM V3

Supported for SELCOPY I/O on FTP Block Mode files.

When FTP is used with "MODE B" to transfer a mainframe data set, allocated as RECFM VB, to a file on a Windows or Unix platform, the BDW at the start of each block is stripped and the 4-byte RDW at the start of each record is replaced with a 3-byte binary record prefix.

The first byte of the prefix contains flag indicators as follow:

x'80'	End of data block is end-of-record.
x'40'	End of data block is end-of-file.
x'20'	Suspected errors in data block.
x'10'	Data block is a restart marker.

The next 2 bytes hold the length of the logical record.

The format of block transmission mode files is documented in detail by the Internet Engineering Task Force RFC #959 for FTP at URL:

http://www.rfc-editor.org/rfc/rfc959.txt

RECFM MFV

RECFM MFV allows input and output of Micro Focus variable length, record sequential files without having to execute calls to the Micro Focus file handler.

These types of file have a header record which is bypassed by SELCOPY's READ operation, so that the first record returned is the first real data record. The header record (length 128) is saved in storage by SELCOPY and made available to the user via POS FHDR.

Example of RECFM MFV input:

```
READ 'microfoc.fil'     RECFM MFV                * Header Record is bypassed.

IF EOF
  THEN PRINT TYPE=D  FROM  POS FHDR  LEN 128    STOPAFT  1    * Header Record.
  ELSE PRINT                         LEN  60    STOPAFT 22    * Data Records.
```

Undefined Record Format

Records are of undefined length. This is type of record organisation is applicable to the following file definitions:

- z/OS and z/VM CMS OS simulated data sets allocated as RECFM U.
- Windows and Unix-like files.

For z/OS and z/VM CMS OS simulated data sets allocated as RECFM U, I/O processing is performed on physical records (blocks) so that the length of a record is the block size. SELCOPY may process a data set of any record format as undefined by specifying RECFM U on the I/O operation.

Undefined record format is default for Windows and Unix-like files, which include STDIN, STDOUT and the Windows clipboard, where end-of-line characters denote the end of a record. The standard end-of-line characters used in a Microsoft Windows environment is carriage-return, line-feed (CRLF) and in a Linux or Unix environment, just line-feed (LF).

SELCOPY supports the EOL parameter to specify the end-of-line characters to be used for RECFM U input and output of Windows and Unix-like files. Supported EOL values are:

EOL	ASCII	EBCDIC	Description
CRLF	X'0D0A'	X'0D15'	Default for Windows, z/OS HFS/ZFS and z/VM BFS.
LF	X'0A'	X'15'	Default for Linux and Unix file systems.
CR	X'0D'	X'0D'	
constant			A quoted or hex character constant of any length.
NO			No end-of-line characters are used.

Unless end-of-line characters are explicitly defined via the EOL parameter, SELCOPY input of Windows or Unix-like files will use any of the supported EOL end-of-line characters (CRLF, LF or CR) as record delimiters. SELCOPY reads a block of data into the input buffer, with buffer size determined by the BLKSIZE value (default 2048), and de-blocks the records using the record delimiters. If no record delimiter is found or EOL NO has been specified, the input record will comprise all data in the input buffer.

Note that end-of-line record delimiters are not included in the record data presented to the program, nor are they included in the input record length value assigned to internal variable, LRECL.

Likewise for RECFM U output, SELCOPY will generate end-of-line characters following each record written unless end-of-line characters are suppressed via EOL NO. The end-of-line characters used will be those defined by the EOL parameter on the WRITE statement. If EOL is not specified, CRLF is used on Windows, LF on Linux and Unix systems and CRNL for z/VM BFS and z/OS HFS/ZFS output.

Chapter 4. SELCOPY Operations

This chapter is a reference to all operations and parameters supported by SELCOPY.

It includes a complete description of each operation's syntax, its classification and the action it performs.

Operation Classification

SELCOPY operations are classified as executing either during control statement analysis or during selection time processing.

Operations executed during control statement analysis are program environment operations, are non-conditional and are executed once only when the statement containing the operation is read by SELCOPY. These operations may be interspersed with selection time operations, although it is recommended that, whenever possible, environment options, variable declarations and equated symbol definitions occur before all other operations.

Specifically, these operations are:

Operation	Description
DECLARE	Declare a variable.
END	Denote the end of control statement input and, if READ CARD exists, the start of in-line data input.
EQU	Define an equated symbol association.
INCLUDE	Nominate the *fileid* of a file containing SELCOPY control statements to be inserted and processed by SELCOPY at the INCLUDE operation location.
OPTION	Specify environment options for the SELCOPY execution.

All other operation and assignment statements are executed at selection time and are subject to the rules of SELCOPY selection time processing (implied loop, etc.) and conditional execution (IF/THEN/ELSE). These operations may be executed any number of times as dictated by the logical flow of processing.

Parameter Specification

In the syntax diagrams for each operation described in this chapter, operation parameter keywords occur in an order whereby related parameters are grouped together. Although these diagrams specify a logical sequence in which parameters may be specificied, it is generally the case that they may be specified in any order. e.g.

```
LOG  STOPAFT=10   REPLY 5 AT 101   'Continue? (Yes/No): '
```

Common Parameters

Some parameters, which relate specifically to selection time processing of an individual statement, are common to all statements assigned a selection identifier. This includes statements that execute a selection time operation (other than IF/AND/OR) or perform a value assignment.

Although referenced in the syntax diagrams of operations within this chapter, with only a few exceptions the affect of these parameters is the same for all operations. Therefore, they are documented separately, with operation specific exceptions included where applicable.

NOPCTL, NOPRINT, NOPSUM

Environment options NOPCTL, NOPRINT and NOPSUM (synonym NOPTOT) are parameters of the OPTION operation. However, they may also be specified as parameters on any other operation or assignment statement, including IF/AND/OR.

These options are used to suppress diagnostic information from being written to the SELCOPY list output. Specifically, NOPCTL will suppress further control statement output, written during control statement analysis, and NOPSUM will suppress summary block output, written at SELCOPY end-of-job. NOPRINT is equivalent to specifying both NOPCTL and NOPSUM.

Note that only the disabling form of these options may be specified on any operation or assignment statement. Their equivalent options to re-activate control statement and summary block output (i.e. PRTCTL and PRTSUM) are not supported in this way.

See the OPTION operation for use of PRTCTL/NOPCTL, PRTSUM/NOPSUM and NOPRINT.

STOPAFT

Applicable on any operation or assignment statement for which a selection identifier has been assigned, the STOPAFT parameter specifies a decimal integer constant value which is the maximum number of times that statement can be selected for execution over the course of the program execution.

SELCOPY maintains a count of the number of times each statement selection is executed. When this count reaches the STOPAFT value for a particular statement selection, that statement is executed for the last time in the current program run before being removed from the collection of statements eligible for execution. e.g.

```
DO  INIT_RTN   STOPAFT=1           * Execute a sub-routine once.
```

Note that the statement selection count includes each repeated execution of that statement caused by a TIMES parameter. e.g. A single execution of an operation with parameter TIMES=10 will increment the execution count for that selection by 10. If both STOPAFT and TIMES are specified on a statement, the STOPAFT value is rounded up to be a multiple of the TIMES value. e.g.

```
READ INDD   TIMES=5   STOPAFT=16   * Process record numbers: 5, 10, 15 and 20.
```

If the STOPAFT values have been reached for all conditional sub-operation (THEN/ELSE) statements subject to the same IF/AND/OR tests, then those tests are also removed from the collection of statements eligible for execution.

Similarly, if the STOPAFT values have been reached for all output operations within the program control statements, then SELCOPY sets immediate end-of-job and the run is terminated. If this occurs when end-of-file has not yet been flagged for the prime input, then SELCOPY ends with return code 4. Furthermore the file sizes of non-VSAM input data objects, recorded in the SELCOPY list summary block, reflect only the number of records read and not the total records in the file.

By default, the maximum number of times that a statement is executed is effectively unlimited. The only exceptions to this are as follow:

Operation	Description
Direct READ	A READ operation that inputs a record from a file using one of the KEY, RBA or REC parameters has a default STOPAFT value of **1** if the key, relative byte address or record number specified is a constant value. e.g. `READ INKS KEY="0001A"`
LOG	A LOG operation has a default STOPAFT value of **50**. This is to prevent excessive, possibly unintentional output being written to STDERR/SYSOUT which, by default, is the user's terminal.
ODBC	An ODBC operation has a default STOPAFT value of **1** if the sql statement is specified as a constant value.
PRINT	For z/OS and z/VM CMS only, a PRINT operation has a default STOPAFT value a set by the CBLNAME option, SPrtStopaft. By default, this option is set to 0 which indicates no limit.

An explicit STOPAFT specified on any statement which executes one of these operations will override the default for that statement only. e.g.

```
LOG  FROM  50 AT 1   STOPAFT 999   * Override the STOPAFT=50 restriction.
```

Synonyms: **STOP S**

TIMES

Applicable on any operation or assignment statement for which a selection identifier has been assigned, the TIMES parameter specifies a decimal integer constant value which is the number of times that statement will be executed when selected. Each execution of the statement increments the statement executions count by 1.

```
LOG  'Error detected...'     TIMES 3      * Output 3 times on each selection.
```

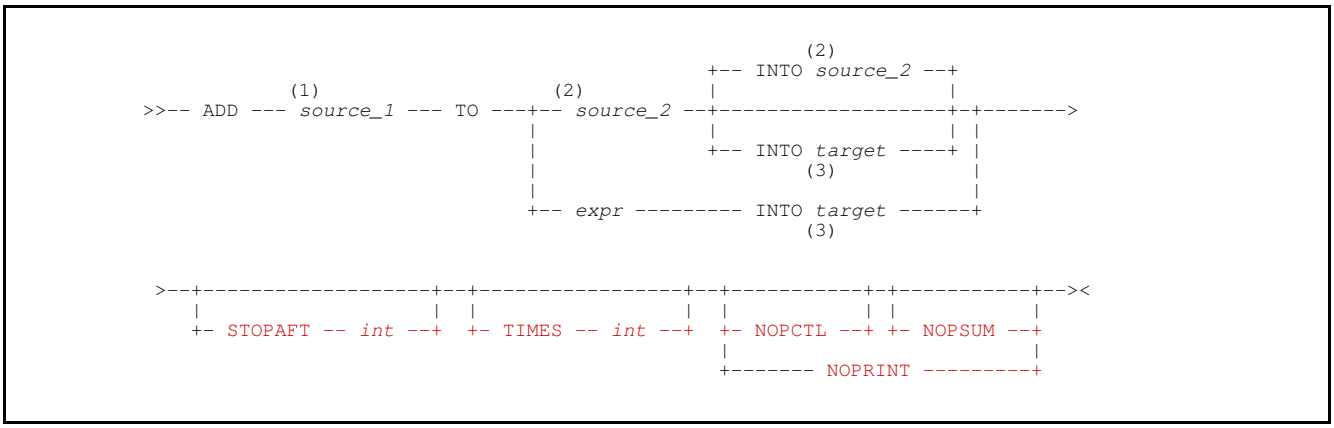
The exception to this is when TIMES is specified on a CHANGE operation. For CHANGE, the TIMES parameter indicates the maximum number of changes that can occur for a successful match of the search string within the source text. The execution count for a CHANGE statement on which TIMES is specified is only ever incremented by 1. e.g.

```
CHANGE   POS 3, LRECL  ':' '#'  TIMES 2   STOPAFT 5   * Executed 5 times max.
```

ADD

Performs an arithmetic addition of two, signed numeric source values, the result of which is assigned to a target variable or stored in a target field.

Syntax:



Syntax Notes:

(1)	***source_1***	must be of numeric or character numeric data type and may be specified as a constant, an arithmetic expression (*expr*), a declared variable or as a field definition of type 2 (*field_p1p2*) or type 3 (*field_nATp*).
(2)	***source_2***	must be of numeric or character numeric data type and may be specified as a declared variable or as a field definition of type 2 (*field_p1p2*) or type 3 (*field_nATp*). If INTO *target* is specified, *source_2* may also be specified as a constant value or an arithmetic expression (*expr*).
(3)	***target***	may be specified as a declared variable or as a field definition of type 2 (*field_p1p2*), type 3 (*field_nATp*) or type 4 (*field_pFMT*).

Description:

Variables and field definitions that constitute the source values and target of the ADD operation may be of different numeric or character numeric data types.

When the source and target values are of different data types and/or if an arithmetic expression or constant value is specified as a source field, then values are converted to a common numeric data type prior to performing the addition. The selected common data type will have a precision and scale which is sufficient to manage the maximum precision and scale of the resultant value.

Any rational number having a fraction value, will be stored internally as a double precision (8-byte) floating point number. Therefore, all fields in the operation will be converted to double precision floating point data type and then floating point arithmetic used.

If a target or source value is expressed as a field definition without a TYPE parameter, the default data type attributed to that field will be the same as the first field definition within the command syntax to have TYPE specified. If no such field definition exists, the default data type is packed decimal integer (TYPE=P). e.g.

```
ADD  4 AT 1 TYPE=C   TO 4 AT 6 TYPE=B   INTO 4 AT 11          * Default TYPE=C
ADD  4 AT 1 TYPE=B   TO 4 AT 6 TYPE=C   INTO 4 AT 11          * Default TYPE=B
ADD  4 AT 1          TO 4 AT 6          INTO 4 AT 11 TYPE=C   * Default TYPE=C
ADD  4 AT 1          TO 4 AT 6          INTO 4 AT 11          * Default TYPE=P
```

Rounding will occur as required on the fractional part of the resultant value based on the number of fractional digits (scale) of the target definition. e.g. If the target variable or field has no scale, the result will be rounded to the nearest integer value by adding 0.5 and dropping the fractional portion.

Parameters:

source_1
: Represents a rational numeric value (addend) that is to be added to the value specified by *source_2*.

TO *source_2*
: Represents a rational numeric value (addend) to which the value specified by *source_1* will be added.

If no *target* is specified, then *source_2* is also the target of the operation. If this is the case, *source_2* must be a declared variable or a field definition.

expr
Represents a numeric or character numeric constant, or an arithmetic expression to be used as the second addend source value.

Note that *source_1* may also be specified in this way. The distinction is made for *expr* and *source_2* because the second source value must also be a valid target if INTO is not specified. Therefore, if *expr* is used, INTO *target* is mandatory.

INTO *target*
Identifies the declared variable or field definition that is the target of the operation. If the second source value is *expr*, then *target* is mandatory.

If required, data conversion and decimal rounding will be performed on the value before it is assigned to *target*.

Default is *source_2* specified on the TO parameter.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Examples:

```
  DECLARE  CELCIUS   DEC(5,2)  INI=37.0         * Declared variable.

  EQU      OREC    1001                        * Output record area.

  OPTION   WORKLEN 2000                        * Work area buffer.

  @VAR        =    4 AT  51 TYPE=B             * Assign @variable value.

  ADD  22     TO   4 AT  51 TYPE=B
  ADD  4.3    TO   8 AT 101 TYPE=P    INTO  8 AT 161 TYPE=F HEX
  ADD  @VAR   TO   CELCIUS            INTO  OREC+12   FMT='S,SS9.99'
```

Return Codes:

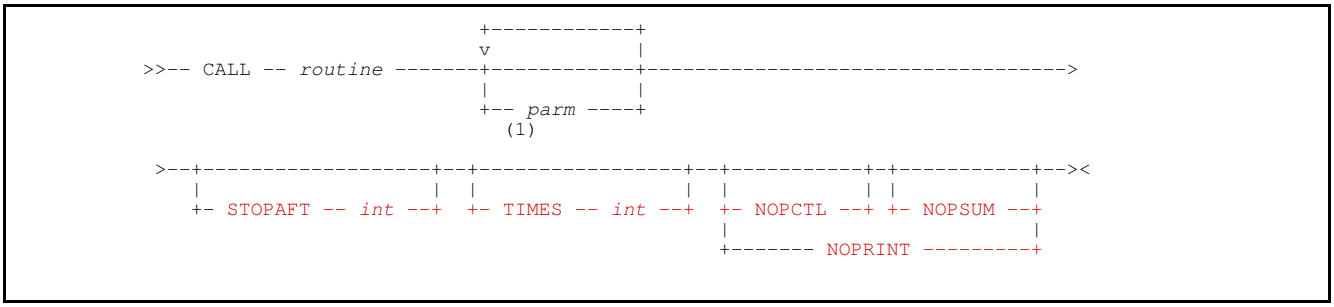
0	Successful completion.
8	One of the following conditions has occurred: 1. *target* has defaulted to *source_2*, which is of data type binary or floating point, and arithmetic overflow has occurred. i.e. A positive value has been added to a positive value *source_2* and the result is negative or a negative value has been added to a negative value in *source_2* and the result is positive. 2. The precision of *target* is not sufficient to contain the resultant value. Truncation has occurred with the loss of significant digits. 3. The first byte of a source or target field definition is within the work area but the last byte is located beyond the end of the work area buffer. The length of the field is reduced so that the last byte of the field is the last byte of the work area buffer. 4. At least one source value is treated as being of packed decimal data type but the source data is invalid packed decimal data.

CALL

Passes control to an external, executable routine (shared object or load module) located in one of the following as applicable to the operating environment:

- A Windows dynamic link library (dll).
- A Unix-style dynamic shared library (so).
- A z/OS load library.

Syntax:



Syntax Notes:

(1) ***parm*** A maximum of 16 parameters may be specified.
parm may be a constant, an arithmetic expression (*expr*) or a declared variable.

Description:

The CALL operation passes control to the named external routine together with a list of parameters using standard linkage as appropriate to the local system in which SELCOPY is executing.

The value returned by the routine must be an integer which is subsequently used to update SELCOPY's return code value (RETCODE) if greater than its current value.

z/OS Systems

On z/OS, the called routine is a load module found first within the standard library search path (STEPLIB, JOBLIB, LPA, Link List). Using standard linkage, the module is passed a single parameter pointer to a list of pointers where the last pointer in this list has the high order bit on.

Unix and Windows Systems

For all other systems, the routine is a shared object found within a shared library which is itself found within the system's library search path. (i.e. the path of directories defined by the environment variable "PATH" for Microsoft Windows, "LIBPATH" for IBM AIX systems and "LD_LIBRARY_PATH" for other Unix systems.) The name of the shared library is established and subsequently opened when SELCOPY encounters the first CALL operation during control statement analysis.

The name of the shared library and the format of the parameter list are based on the prevailing value for the CALLTYPE environment option.

CALLTYPE DIRECT

For CALLTYPE DIRECT (the default and recommended value), the named external routine (*routine*) is called directly as a function within a shared library with a single parameter pointer to an array of character string pointers. Thus, every called routine must be defined as:

```
int  routine (char **parms);
```

The shared library name for CALLTYPE DIRECT, depends on the prevailing value for the LIBNAME environment option when the first CALL is identified and the library opened. Note that any change in the value for option LIBNAME that occurs on a statement processed following the first CALL operation will have no effect. This is because SELCOPY will not perform another library open for subsequent CALL operations.

If a value has been specified for LIBNAME, the named shared library will be used. Otherwise, the default library name is **libselc.dll** for Windows and **libselc.so** for Unix systems.

For further details on routine linkage, endianness and compile and link requirements, please refer to the sample C/C++ header (slccall.h) and source (slccall.c) files provided with the product install material.

CALLTYPE VIA_SLCCALL

For CALLTYPE VIA_SLCCALL, the function ***slccall*** is called with two parameter character pointers, the first a pointer to the function name to be executed (*routine*) and the second a pointer to an array of character string pointers to ultimately be passed to *routine*. The controlling *slccall* routine needs to be adapted so that it tests the name of *routine* and then calls a function of the same name with a parameter equal to the second parameter pointer passed to slccall. e.g.

```
int  slccall  (char *routine,  char **list )
{
 if      (strcmp(nam, "routine" ) == 0)     return  routine(list);
 return -1;      // Negative return value to indicate routine not found.
```

The *slccall* routine must exist in a shared library named **slccall.dll** for Windows, **libselc.sl** for HP-UX Unix and **slccall.so** for all other Unix systems.

The linkage provided by option CALLTYPE VIA_SLCCALL exists only to support routines written for CALL operations executed by releases of SELCOPY prior to version 2.08 build 902. It is recommended that this type of linkage no longer be used.

Parameters:

routine

The name of an external routine to which SELCOPY will pass control.

For z/OS, this is an up to 8 character load module name. For all other systems, routine is the name of a shared object belonging to a Unix dynamic shared library (.so) or Windows dynamic link library (.dll).

parm

An input parameter of numeric integer or character data type that will be passed to the called routine. Each parameter is referenced in the external routine as a pointer to (the address of) an area of storage.

If the parameter is numeric or character numeric, then 1 is subtracted from the value and it is used as an offset from the base storage address (i.e. position 1 of the work area or input record buffer). Thus, a parameter pointer for a numeric value will address the area of storage at this offsetted position. e.g. parameter value 1 points at position 1 of the work area.

If the parameter value is of character data type, the pointer will be the address of the character text. For declared variables, this will be the address of the character variable's source field. For character constants, it will be an address of an area of storage containing the null terminated constant value.

If the source field of a declared variable is passed as a parameter, the pointer will be the source field address. This is the method by which the current value of a numeric or character numeric variable is passed to the external routine.

NOPCTL, NOPSUM, NOPRINT

See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*

See common parameter STOPAFT for details.

TIMES *int*

See common parameter TIMES for details.

Examples:

z/OS Assembler Routine

The following simple Assembler source is assembled and link edited to create a load module CALLTEST. The routine uses the character addressed by the first parameter to replace each character at positions in storage addressed by subsequent parameters.

```
CALLTEST CSECT
     USING *,15
     STM   14,12,12(13)        Save registers.
     L     2,0(1)              1st Parm, Source.
     IC    3,0(2)
LOOP LTR   2,2                 Test the flag bit which makes value negative.
     BM    EXIT                If we've just done last one. (Branch Minus)
     LA    1,4(1)              Next Parm pointer, Destination.
     L     2,0(1)
     STC   3,0(2)
     B     LOOP
EXIT LM    14,12,12(13)        Restore registers.
     LA    15,243              Set a Return Code, other than 0.
     BR    14
     END
```

The following SELCOPY program calls routine CALLTEST.

```
DECLARE  BPOS    BIN   INI=18   * Variable integer value.

OPTION  WORKLEN 100             * Work area auto initialised to blanks.
PRINT                           * Print a blank line length 80.

CALL  CALLTEST   'X'   BPOS  5    10    15   20   25
                                * Will also set Retcode=243.
PRINT                           * Should have "X" in pos 18,5,10,15,20 and 25.

IF POS 26, 33  = 'X'            * Should succeed at pos 30 and set @ ptr.
  THEN CALL CALLTEST  X'E9'  @+8  @+11  LRECL-32      * (X'E9' is 'Z')

PRINT                           * Should now have Z in pos 38,41 and 48.
```

Unix or Windows C++ Routine

The following simple C source function, dupstr, is compiled and linked as a shared object in library libselc.so. The routine copies the null terminated character string pointed to by the first parameter, to positions in storage addressed by the remaining pointers in the parameter list.

```
int  dupstr  (char **list)
{
 int i;
 int len;
 char *src;                                 /* Source data elem. */

  if (!list[0])    return 0;                /* Do nothing if no parms. */
  src = list[0];                            /* Source data. */
  len = strlen(src);
  for (i=1; list[i]!=NULL; i++)
     memcpy (list[i], src, len );           /* Duplicate a string. */
 return 247;
}
```

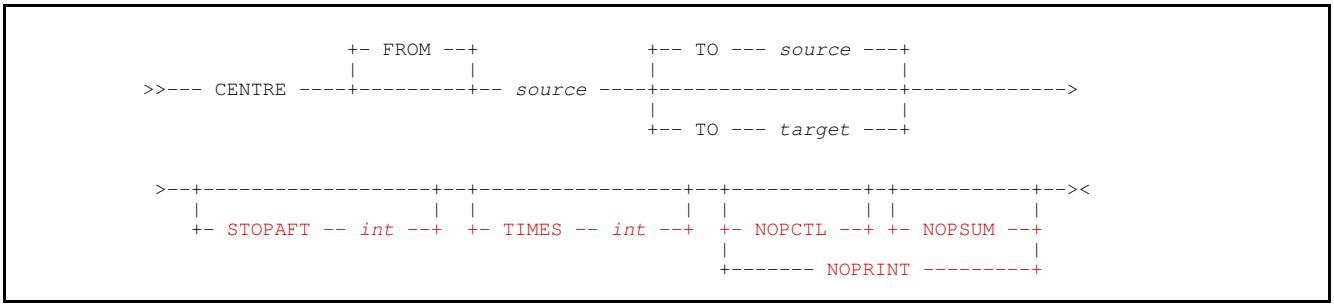
The following SELCOPY program calls routine dupstr.

```
DECLARE  SOURCE  CHA(20)  INI="Hello World"
DECLARE  DEST1   BIN      INI=8

OPTION  WORKLEN 100             * Work area auto initialised to blanks.
PRINT                           * Print a blank line length 80.

CALL  dupstr     SOURCE    DEST1   29
                                * Will also set Retcode=247.
PRINT                           * "Hello World" in pos 8 and 29.
```

CENTRE

Centralise text in a character source value.

Syntax:



Synonyms:

CENTRE	CENTER ADJC

Description:

The text represented by the *source* character value is centralised and assigned to the *target* value. The *target* location may overlap storage at the *source* value location and if *target* is not specified, then the *source* value is updated.

The source text value is centralised in an area of storage containing all blank characters and of length equal to the *target* value field area if specified, otherwise the length of the *source* value field area.

```
DECLARE  CVAR   CHAR(20)  INI='Hello'
CENTRE   CVAR                        *          ....,....1....,....2....,
PRINT    CVAR  'World'               * Prints: "        Hello       World"
```

Multiple consecutive blank characters that exist between non-blank characters in the source value text, are condensed to a single blank character in the centralised value. e.g.

```
POS     101 = "xxx      yyy zz"      *          ....,....1....,....2
CENTRE  15 AT 101   TO  POS 1, 20    * To get: "     xxx yyy zz     "
```

Parameters:

FROM *source*
References the character text value to be centralised. *source* may be specified as a declared variable of character data type, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition.

If TO *target* is specified, *source* may also be specified as a quoted or hex character constant.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *target* is implied. Otherwise, ERROR 69 is returned.

TO *target*
Identifies the declared variable or Type 1, 2 or 3 field definition that is the target of the operation. If *source* is a constant value, then TO *target* is mandatory.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *source* is implied.

Default is TO *source*.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

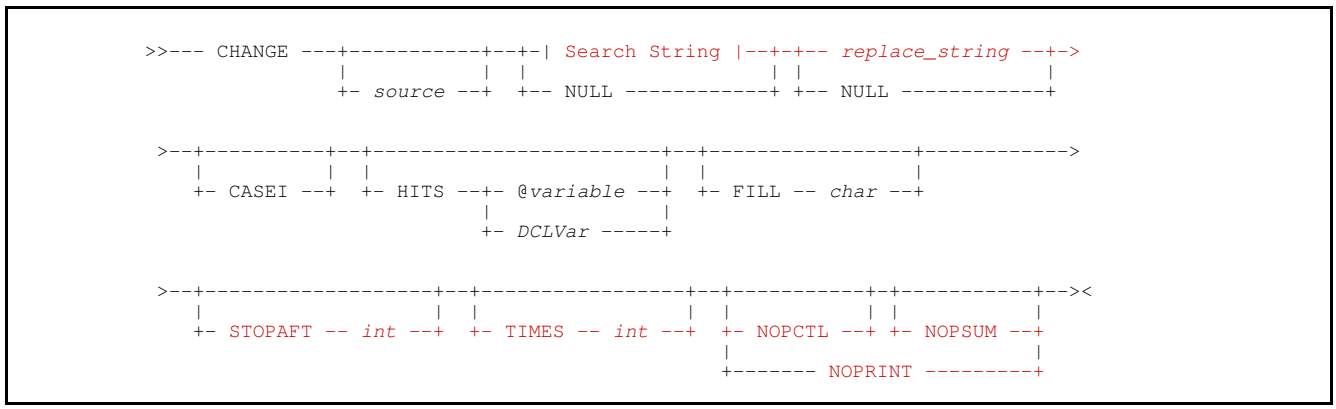
STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of CENTRE for the same source and target values is unnecessary and so TIMES should never be used on the CENTRE operation. See common parameter TIMES for details.

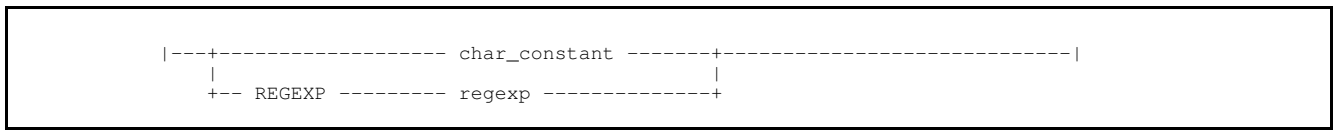
CHANGE

Change occurrences of a text string value in a character source.

Syntax:



Search String:



Synonyms:

CHANGE	C CH CHG

Description:

Operating on character data only, the CHANGE operation will scan the *source* text value for the every occurrence of a specified search string and replace the matching text with the specified replacement string (*replace_string*).

Using the TIMES parameter, the operation may be restricted to a limited number of occurrences of the search string that occur first within the source text. Following a successful search and replace, the next, repeated execution of the operation begins the scan for the search string at the character following the replaced text. i.e. occurrences of the search string within the replaced text will not be included in the scan. e.g.

```
POS 1 =                                          "ABCDEFG ABCDEFG ABCDEFG"
CHANGE  80 AT 1    "ABC"  "abc"  TIMES 2   * To get: "abcDEFG abcDEFG ABCDEFG"
                                            *         (only first 2 occurrences.)

CHANGE  80 AT 1    "EFG"  "EFG#" TIMES 2   * To get: "abcDEFG# abcDEFG# ABCDEFG"
CHANGE  80 AT 1    "BCD"  "bcd"  TIMES 1   * To get: "abcDEFG# abcDEFG# AbcdEFG"
```

Regular expressions may be used to specify the search string value. Furthermore, tagged sub-expressions in the search string, may be used to identify text patterns that are to be included in the *replace_string*. A tagged sub-expression forms part of the regular expression pattern string and is defined by enclosing a regular expression definition within pairs of "{ }" (braces). The *n*th tagged sub-expression in the search string may be referenced by &*n* in positions of the search string and/or *replace_string* that follow. When a match is found for the search string the text matching the tagged sub-expression *n* is substituted for each occurrence of &*n* that follows. e.g.

```
DECLARE  WK1    CHA (26)                    INI "AB<b>ACAC</b>"
CHANGE   WK1 RGX '{A?}&1'  '--&1--'    * Gives: "AB<b>--AC--</b>"
```

The special tagged sub-expression reference, &0, treats the entire regular expression pattern search string specification as a tagged sub-expression. Therefore, it may be used in *replace_string* to duplicate text matched by the search string. e.g.

```
DECLARE  WK1    CHA (26)                    INI "A<b>BACAC</b>"
CHANGE   WK1 RGX 'B?@'       '&0&0'    * Gives: "A<b>BACACBACAC</b>"
```

If the length of the replacement text is less than that of the text matching the search string, then characters within the *source* value that follow the replaced text, are shifted left. Unless *source* is a variable length character declared variable, then it is padded on the right using the character specified by the FILL parameter.

If the length of the replaced text is greater than that of the text matching the search string, then characters within the *source* value that follow the replaced text, are shifted right and truncated. The replaced text may itself be truncated if it extends beyond the

length of the *source* text. SELCOPY return code 8 is set if a truncated character does not match the pad character specified by the FILL parameter (default blank).

Parameters:

source
References the character value in which text will be changed. *source* may be specified as a declared variable of character data type, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition.

If *source* not specified, *source* defaults to be a field definition starting at position 1 with length equal to the prevailing value of the variable LRECL. (i.e. FROM POS 1 LENGTH LRECL).

Search String | **NULL** | **NUL**
Specifies the search text string for which the *source* value will be scanned. It may be specified as one of the following:

char_constant
A quoted or hexadecimal character constant.

REGEXP *regexp*
REGX
RGX
A regular expression.

Note that, if a regular expression search string contains tagged sub-expressions, then the text matched by these sub-expressions may be substituted in the text of the replace string using tagged sub-expression references (&*n*).

The keyword, NULL or NUL, may be specified to indicate a null (length zero) string value for the search string. Use of a null search string will force a default value of 1 for parameter TIMES.

The first occurrence of a NULL search string is found immediately before the first character of the *source* value.

replace_string | **NULL** | **NUL**
Specifies the replacement text string used to replace text that matches the search string. The *replace_string* may be specified as a quoted or hex character constant. If the search string is a regular expression, then *replace_string* may contain tagged sub-expression references, (&*n*).

The keyword, NULL or NUL, may be specified to indicate a null string value for *replace_string*. Use of a null replace string will remove occurrences of text matching search string from *source*.

An error message is given if a NULL keyword is used in place of both the search string and *replace_string*.

CASEI
Indicates that the scan for the search string is case insensitive. e.g.

```
DECLARE str1 CHAR                    INIT="ABCDEFG ABCDEFG ABCDEFG ABCDEFG"
CHANGE  str1  'abc' 'zz ' CASEI  * To get: "zz DEFG zz DEFG zz DEFG zz DEFG"
CHANGE  str1  'efg' '.'   CASEI  * To get: "zz D. zz D. zz D. zz D.        "
```

FILL *char*
PAD
Defines the pad character (*char*) to be used when the length of the replacement text is less than the length of the text matching the search string. The pad *char* may be specified as a quoted or hex character constant of length 1.

It also defines that character that may be truncated without setting SELCOPY return code 8. Truncation occurs when the length of replacement text is greater than the text matching the search string.

The default is the blank character.

HITS *@Variable* | *DCLVar*
HITS nominates an *@variable* or declared variable (*DCLvar*) of numeric data type. Following execution of the CHANGE operation, a value will be assigned to *@variable* or *DCLvar* which is equal to the number of occurrences of the search string that have been replaced in the source text.

If the search string is not found, the value is 0 (zero).

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Unless the search string is NULL, TIMES restricts the number of occurrences (*int*) of the search string within the source text for which the text will be replaced. The value *int* must be specified as a decimal integer constant.

If the search string is NULL, TIMES *int* specifies the number of times the CHANGE operation will be executed.

If NULL is specified in place of a search string, the default is TIMES 1. Otherwise, every occurrence of the search string found in *source* is replaced. Also see common parameter TIMES.

Example:

The following working example demonstrates basic use of the CHANGE operation.

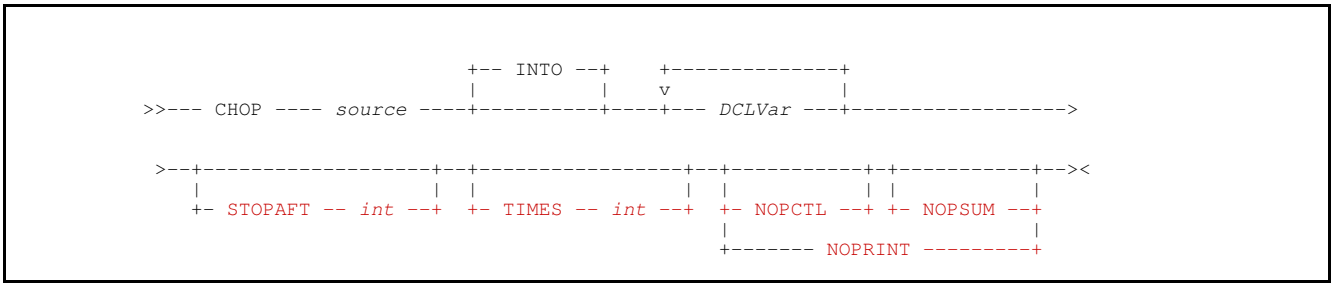
```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2015/11/27 18:09   PAGE   1
------------------------------------------------------             ----------------   --------


          equ wid          66                  * For width of data on a PRINT line.
          option pagewidth=94  datawidth=wid    * Worklen=80 provided by default.

                             * ....,....1....,....2....,....3....,....4....,....5....,....6....,.
          dcl a_z      cha ini 'ABCDEFGHIJKLMNOPQRSTUVWXYZ ### and ### abcdefghijklmnopqrstuvwxyz '
          dcl nums     cha ini 'ABCDEFG  11,222.33  4,555.66  77,888.99  111,222.33  44,555.66    '
          dcl xxx      cha ini 'xxxxxxxxxx   abc    xxxxxxxxxx    def   xxxxxxxxxx    ghi   xxxxxx'

     1.   line 1               stopaft=1
     2.   lrecl = wid

     3.   pos 1 =        a_z
     4.     then print                                                 * Original data.
     5.     then change  a_z  '###'  '-'   times=3  hits=@hits fill='X'    * source is a DCLVar.
     6.     then pos 1 = a_z                          * Copy changed data to POS 1  LEN=66.
     7.     then do showit   '/###/  to  /-/'

     8.   pos 1 = nums
     9.     then print                                                 * Original data.
    10.     then change       ','    NULL  times=22 hits=@hits             * source is default.
    11.     then do showit   '/,/    to  NULL'
    12.     then change      ' '   '   '  times=2  hits=@hits stopaft=1   * 2 blanks to 3,  2 times.
    13.     then do showit   '2 blanks to 3 blanks   times 2 '

    14.   pos 1 = xxx
    15.     then print                                                 * Original data.
    16.     then change       'x'    NULL          hits=@hits pad='Z'     * Omit TIMES.
    17.     then print   '@hits = '  @hits fmt=99   "   changing  /x/   to   NULL   with fill='Z'"
    18.     then space 1
    19.     then change       NULL   '---#'        hits=@hits             * RC=8 expected.
    20.     then do showit   "NULL     to   /---#/  RC=8 as ZZZZ trunc'd."

    21.   eoj

         =showit:=    desc                                * The ":" indicates params exist.
          ------
    22.   print '@hits = '  @hits fmt=99   '   changing  '   desc
    23.   print                                                       * Changed data.
    24.   space 2
    25.  =return=


SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2015/11/27 18:09   PAGE   2
------------------------------------------------------             ----------------   --------
    INPUT   SEL SEL                                                                    RECORD
    RECNO   TOT ID.          1         2         3         4         5         6       LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,. ------
        0     1   4 ABCDEFGHIJKLMNOPQRSTUVWXYZ ### and ### abcdefghijklmnopqrstuvwxyz      66
        0     1  22 @hits = 02   changing  /###/  to  /-/                                66
        0     1  23 ABCDEFGHIJKLMNOPQRSTUVWXYZ - and - abcdefghijklmnopqrstuvwxyz XXXX     66


        0     1   9 ABCDEFG  11,222.33  4,555.66  77,888.99  111,222.33  44,555.66        66
        0     2  22 @hits = 05   changing  /,/    to  NULL                               66
        0     2  23 ABCDEFG  11222.33  4555.66  77888.99  111222.33  44555.66            66


        0     3  22 @hits = 02   changing  2 blanks to 3 blanks   times 2                66
        0     3  23 ABCDEFG   11222.33   4555.66  77888.99  111222.33  44555.66          66


        0     1  15 xxxxxxxxxx   abc    xxxxxxxxxx    def   xxxxxxxxxx    ghi   xxxxxx     66
        0     1  17 @hits = 36   changing  /x/   to   NULL   with fill='Z'                66

        0     4  22 @hits = 01   changing  NULL     to   /---#/  RC=8 as ZZZZ trunc'd.     66
        0     4  23 ---#   abc        def       ghi   ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ     66


                   ....,....1....,....2....,....3....,....4....,....5....,....6....,.

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL          FSIZE   CI    DSN
 ------      ------      ----     -------  -----          -----   --    ---
    1---18        1
   19             1      (***01 RETCD=8***)
   20---21        1

                  =showit=
   22---25        4 =ret=

***WARNING*** (SEL---19)        8 = RETURN CODE FROM SELCOPY

    ** SELCOPY/WNT 3.30.002  2015/10/05  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 2. The CHANGE operation.

CHOP

Chop a character source value into a number of values to be assigned to SELCOPY declared variables.

Syntax:



Synonyms:

CHOP	PARSE

Description:

The CHOP operation will parse a source character value so that strings of non-blank characters each constitute a value to be assigned to a declared variable. Each non-blank character string is delimited by either one or more consecutive blank characters or the end of the source value.

Parsing the source value from left to right, each parsed value is assigned to the next available declared variable following the source specification. i.e. The first parsed value is assigned to the first declared variable following *source*, the second value to the second variable, etc.

If there are more parsed values than variables, then the *source* is parsed as normal for all but the last variable. The last variable is assigned a value equal to the remainder of the *source* value with leading and trailing blanks removed. If there are fewer parsed values than variables, then the extra variables are assigned a blank value if of character data type, and a value of 0 (zero) if numeric or numeric character.

If a declared variable is specified which is of numeric or numeric character data type, then its equivalent parsed value is treated as being of numeric character data type. The numeric value it represents is assigned to the variable.

On assigning a parsed value to a variable of character data type, SELCOPY return code 8 is set and the value truncated if its length is greater than the maximum defined for the variable.

Parameters:

source
References the character text value to be parsed. *source* may be specified as a quoted character constant, a declared variable of character data type, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition.

INTO *DCLVar*
Identifies a declared variable (*DCLVar*) to which a parsed value will be assigned. The data type of the variable determines the interpretation (character or numeric) of the parsed value.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of CHOP for the same source value is unnecessary and so TIMES should never be used on the CHOP operation. See common parameter TIMES for details.

Example:

The following example demonstrates parse of a source character string, first as character values into fixed length character variables, then as numeric values into variables of numeric data type.

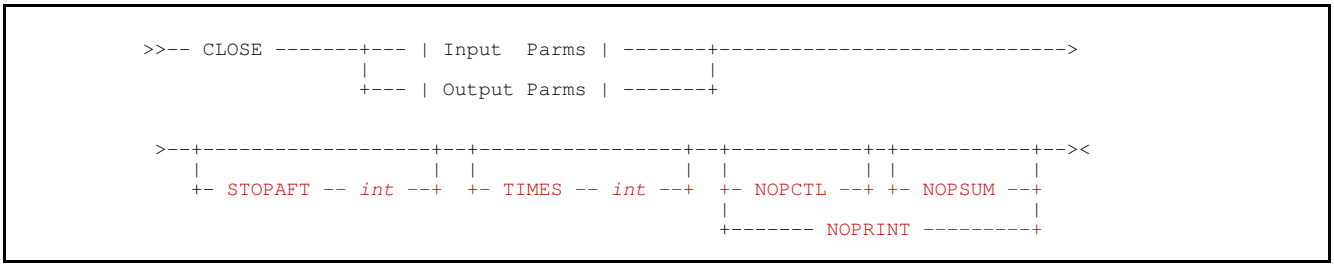
```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2015/11/30 14:41   PAGE   1
-----------------------------------------------------              ----------------   --------


          declare src   char( 60  )   ini='110119   1,234,567.85   555,666.9999777  .065'

          declare ch1   char(  9  )
          declare ch2   char( 15  )
          declare ch3   char( 20  )
          declare ch4   char( 10  )

          declare v1    bin           fmt=           's,sss,ss9'
          declare v2    flt (  8  )   fmt=        '   z,zzz,zz9.99'
          declare v3    dec ( 31,4)   fmt='         zzz,zzz,zz9'
          declare v4    flt (  8  )   fmt=          'z,zzz.zzzz'

          option    datawidth=90

     1.   chop  src   into          ch1           ch2           ch3           ch4
     2.   print           '  ch1="' ch1 '"  ch2="' ch2 '"  ch3="' ch3 '"  ch4="' ch4 '"'

     3.   chop  src   into           v1            v2            v3            v4
     4.   print           '   v1="'  v1 '"   v2="'  v2 '"   v3="'  v3 '"   v4="'  v4 '"'


    INPUT   SEL SEL                                                                                 RECORD
    RECNO   TOT ID.          1         2         3         4         5         6         7         8         9  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
        0     1   2   ch1="110119   "  ch2="1,234,567.85   "  ch3="555,666.9999777     "  ch4=".065      "          80
        0     1   4    v1=" +110,119"   v2="   1,234,567.85"   v3="             555,667"   v4="    0.0650"          80
                    ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1----4        1


    ** SELCOPY/WNT 3.30.002  2015/10/05  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                  ** Expiry: 14 Jul 2016 **
```

Figure 3. The CHOP operation.

CLOSE

Explicitly close a data object currently open for input or output.

Syntax:



Description:

The CLOSE operation will close a data object associated with the specified or implied *fname*, which is currently open for input or output. If the object is already closed, then no action is taken.

By default, data objects that have been opened during execution of a SELCOPY program, are automatically closed by SELCOPY at end of job. The CLOSE operation is only required if a data object is to be re-opened or an *fname* is to be associated with another object during the course of the program execution.

If a CLOSE operation exists for *fname* and, for input objects only, its control statement occurs before one that contains a READ operation for the same *fname*, then the open of the data object will be deferred until a READ, WRITE or OPEN is executed for *fname* during selection time processing.

Following execution of CLOSE, the data object associated with *fname* will remain closed until another READ, WRITE or OPEN operation is executed for *fname*, at which point the appropriate open for input, input update or output will be performed for the object. If the *fileid*, *list_command*, *table_name* or *full_select* referenced by the I/O operation is specified as a variable name or field definition, then the data object is dynamic. In this case the object associated with *fname*, and therefore opened when the I/O operation is executed, is determined by the prevailing value assigned to the variable or field definition. Note that, a change to the value assigned to these variable sources will have no affect on the object associated with *fname* unless a subsequent open is to be performed.

Note that execution of an OPEN operation will first perform a close of the object assigned to *fname* if it is already open. The difference between CLOSE and OPEN being that OPEN will immediately re-open the object associated with *fname* having executed the close.

If no work area is defined and input data is read into position 1 (the default), then following close of an input data object, position 1 will be the start of a blank record of length 80.

CLOSE may be particularly useful if a file object needs to be re-opened using a different I/O method within the same SELCOPY program. e.g. Records may be written to a file and, when finished, the file may be closed and then read (using a different *fname*). See example 2. below.

Parameters specified on the CLOSE operation are those supported by a READ operation for open or data retrieval, or those supported by a WRITE operation for open or data output. Unless overridden by a subsequent READ or WRITE operation on *fname*, these parameters apply to all subsequent I/O operations on *fname*.

Parameters:

`Input Parms`
: In addition to a specified or implicit *fname*, CLOSE for an input data object supports any of the **Object Open** or **Data Retrieval** parameters supported by the READ operation as follow.

fileid	EOL	RAW	SSN
BDW / NOBDW	ESDS	RDW / NORDW	SUBDIR
BLKSIZE	FILL	RECFM	TABS
DEFER	FMT	RRDS	TABLE
DIR	HEADER	SELECT	UPD
DIRDATA	KSDS	SEP	USER
DIRTYPE	LIST	SORT	VSAM
DSN	LRECL	SORTDIR	WHERE
DSNPFX	ODBCPASS	SQL	

See operation READ for details on parameter usage and descriptions.

`Output Parms`
In addition to a specified or implicit *fname*, CLOSE for an output data object supports any of the **Object Open** or **Data Output** parameters supported by the WRITE operation as follow.

fileid	EOL	LRECL	TABLE
APPEND	ESDS	ODBCPASS	TRUNC / NOTRUNC
BDW / NOBDW	FMT	RDW / NORDW	USER
BLKSIZE	FILL	RECFM	VSAM
DEFER	KEYLEN	REUSE	WIN
DSN	KEYPOS	RRDS	
DSNPFX	KSDS	SSN	

See operation WRITE for details on parameter usage and descriptions.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of CLOSE for the same data object is unnecessary and so TIMES should never be used on CLOSE. See common parameter TIMES for details.

Examples:

Example 1.

In the following example, card records (which follow the END operation) are read from the same control statement input file. Each card record contains the z/OS DSN of a data set for which records are to be scanned for 21st century ISO dates (e.g. 2009/04/12).

The CLOSE operation is used to close the data set associated with INDD once end-of-file has been detected. The next time the READ operation is executed for INDD, the DSN currently assigned to variable IDSN will be associated with INDD and opened ready for input.

```
  DECLARE IDSN   NTS(54)                          * Null terminated char string.
  DECLARE IREC VCHAR(32760)                       * Variable length char string.

  READ   CARD            INTO IDSN                * Input the new DSN.
  PRINT '*** INPUT DATA SET: '    IDSN            * Print the new DSN.
  DO     CHECKDATE                                * Process the data set.
  SPACE 2                                         * Print 2 blank lines.
  GOTO GET                                        * Get the next CARD input.


==CHECKDATE==     ** Print records that have a 21st century data pattern. **
  READ   INDD  DSN=IDSN  INTO IREC                * Input a record from DSN.

  IF EOF       INDD                               * If end-of-file.
    THEN CLOSE INDD          NORDW                * Close data set so that INDD
                                                  * can be re-opened as new DSN.
    THEN GOTO  CHECKDATE_END                      * End sub-routine processing.

  IF IREC = RGX '20:d^2\/:d^2\/:d^2 '  PTR=@DATE  * Scan data for date pattern.
    THEN PRINT TYPE=C  FROM @DATE                 * Print the date to verify.

  GOTO CHECKDATE                                  * Process the next record.

 =CHECKDATE_END=
  RETURN                                          * Return to main process loop.

  END
NBJ.£ABC
NBJ.CBLINST.CBL12075.JGE.SELCOPYI.CMX
CBL.JCL(CBLILNKD)
```

Example 2.

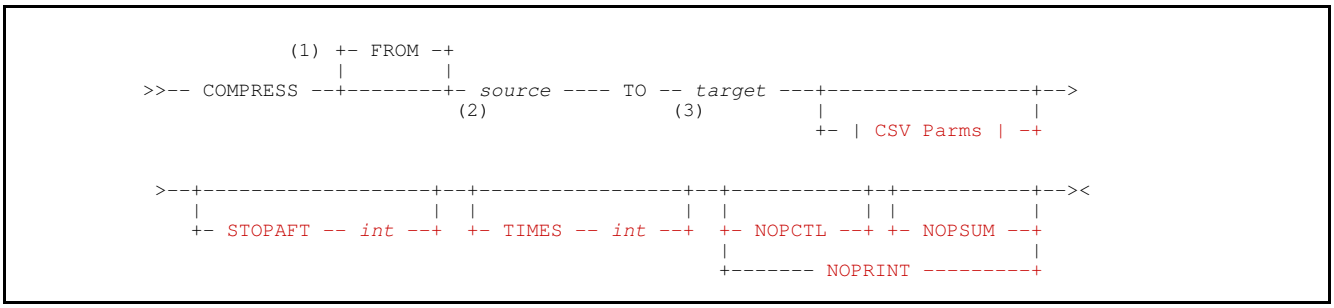
In the following example, records are copied to an output file. Once all records have been copied, the output file is closed and then re-opened for input. The copied records are read and printed.

```
  READ         INPF DSN='NBJ.SSCLOS02.DATA'      NORDW       * Read source file.
  IF EOF       INPF                               * If end-of-file.
    THEN CLOSE OUTF DSN='NBJ.SSCLOS02.DATA.COPY'  * Close the output file.
    THEN DO    PRINT_COPY                         * Process the sub-routine.
    THEN EOJ                                      * End-of-job.

  WRITE        OUTF                               * Write the output record.
  GOTO GET                                        * Get the next input record.


==PRINT_COPY==    ** Print records written to the copy file. **
  READ         COPF DSN='NBJ.SSCLOS02.DATA.COPY' NORDW DEFER * Input deferred.

  IF EOF       COPF                               * If end-of-file.
    THEN GOTO  PRINT_COPY_END                     * End sub-routine processing.

  PRINT                                           * Print the copied record.
  GOTO         PRINT_COPY                         * Get the next copied record.

 =PRINT_COPY_END=
  RETURN                                          * Return to main process loop.
```

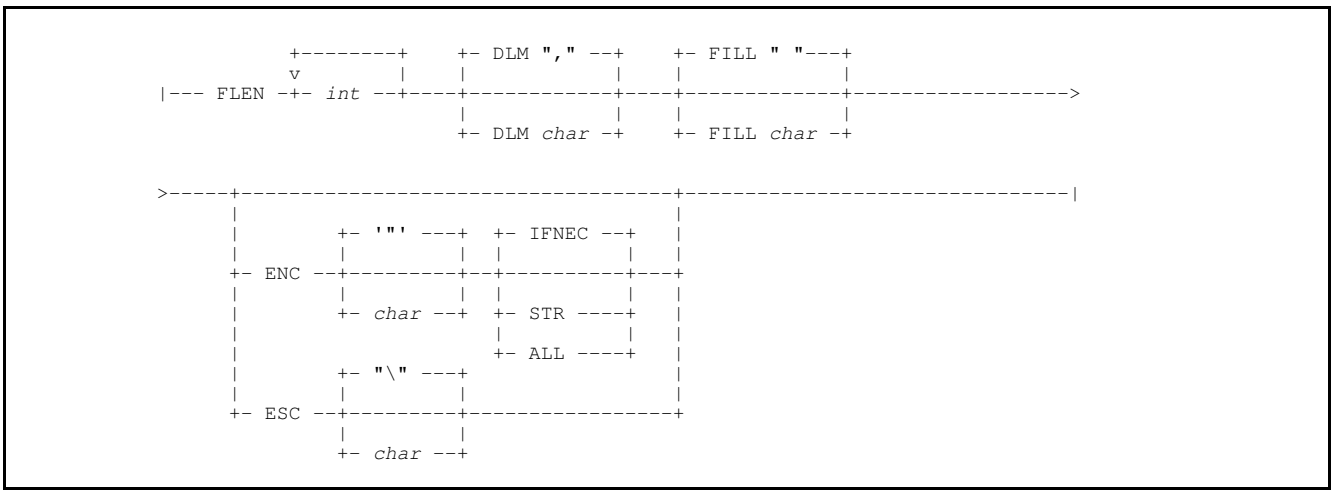
COMPRESS

Compress a line of data using proprietary compression algorithms or compress data in fixed field positions to a comma separated variable (CSV) format.

Syntax:



CSV Parms:



Syntax Notes:

(1)	**FROM**	is mandatory if *source* is a field definition of type 1 (*field_pLENn*) but with keyword POS omitted.
(2)	***source***	references an area of storage specified by a character constant, a declared variable of character data type, or as a field definition of type 1 (*field_pLENn*), type 2 (*field_p1p2*) or type 3 (*field_nATp*).
(3)	***target***	references an area of storage specified by a field definition of type 1 (*field_pLENn*), type 2 (*field_p1p2*) or type 3 (*field_nATp*).

Description:

COMPRESS and EXPAND operate on single lines of data (usually file object input records) to perform one of the following tasks:

1. Compress or expand data using SELCOPY's proprietary compression algorithms.
2. Compress or expand values in comma separated format (CSV).

For both types of data compression, following execution of the COMPRESS operation the value assigned to internal variable LRECL is updated to be the length of the compressed data. Furthermore, the storage containing the *source* data must not overlap the *target* storage area.

Specification of a work area is mandatory for COMPRESS.

Data Compression

Storing file data on fixed storage devices may be inefficient, particularly if a file object contains a large number of records that are long and consist of many embedded blanks and other duplicated characters.

In addition to the hardware and software compression options available on most systems, the COMPRESS/EXPAND operations may be used to invoke SELCOPY's own proprietary compression algorithms for record data read from or written to a file object.

It should be noted that, if the data being compressed is short and does not contain long chains of repeating characters, then it is possible that the compressed data is longer than the original source.

CSV Data Processing

Data within database tables and spreadsheets can usually be unloaded into a comma separated variable (CSV) format, where each column value is separated from the following column value by a delimiter character, usually a "," (comma). In most processing environments, unloaded database data is the most common source of CSV data.

SELCOPY can also generate CSV format output. Using the COMPRESS operation, data at consecutive fixed positions may be compressed to create a variable length CSV record in storage.

SELCOPY's CSV data compression operates on values that are arranged at consecutive fixed locations within the *source* area. These fixed locations are usually, though not exclusively, determined by an EXPAND operation which has previously been executed on the CSV data.

Parameter FLEN must be specified in order to invoke CSV data compression instead of SELCOPY's general data compression utility. The FLEN parameter values identify the fixed lengths of each storage area at which the data values are located. When compression occurs, each data value is copied to the next available position in the *target* area with trailing characters, as specified by the FILL parameter, omitted. Provided the *source* data area still contains unprocessed values, the value delimiter character is appended to the value copied to the *target* area.

By default, the source data values are not checked for blanks or special characters other than the delimiter character as defined by parameter DLM. If ENC and ESC parameters are both omitted, an error is returned if the source data already contains the delimiter character. If this occurs, nothing is copied to the target area and internal variable LRECL is set to 0 (zero).

Based on individual characters that exist within the source data, conditional use of enclosing characters (e.g. quotation marks) or escape characters may be invoked using the ENC or ESC parameters respectively.

Warning: Truncation will not occur when the last value copied to the target area exceeds the target area length. However, an error will occur if the first character of any of the values falls outside the target area.

Parameters:

FROM *source*

Specifies a character value or area of storage containing the source data to be compressed. *source* may be specified as a character constant, a declared variable of character data type, or as a field definition of type 1 (*field_pLENn*), type 2 (*field_p1p2*) or type 3 (*field_nATp*).

The complete length of character data represented by *source* will be compressed. If FLEN is specified to indicate CSV data compression and the length of *source* is greater than the sum of the specified FLEN values, then the last FLEN value will be used to define fields in the surplus *source* data. (See FLEN below.)

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL.

Keyword FROM need not be specified if *source* is specified as a character constant or a declared variable.

TO *target*

Specifies the area of storage into which the compressed data will be moved. *target* may be specified by a field definition of type 1 (*field_pLENn*), type 2 (*field_p1p2*) or type 3 (*field_nATp*).

If specified as a type 1 field definition but without a length, the field will have a default length equal to that of *source*.

FLEN *int ...*

Identifies the COMPRESS operation as being for CSV data generation. FLEN specifies an integer constant value, *int*, or a sequence of integer constants that correspond, one-to-one with the lengths of each consecutive, fixed length data area in *source*.

If the sum of the integer values specified by FLEN is lower than the length of the *source* specification, then the last FLEN value in the sequence is used to identify the lengths of all remaining data fields in *source*. e.g. `FLEN 30 20` indicates that the data area containing the first value is of length 30 and data areas containing all remaining values are length 20.

If the sum of the FLEN values is greater than the length of the *source* area, then extraneous values are ignored.

DLM *char*
DELIM

Specifies the delimiter character *char*, a character constant of length 1, used to separate each value in the compressed CSV output.

Default is "," (comma).

FILL *char*
PAD
Specifies the fill character *char*, a character constant of length 1, used to pad short values within the fixed length source data areas.

The COMPRESS operation will strip consecutive occurrences of this character that exist at the end of the source data areas so that they are not included in the value copied to the *target* area.

Default is "b" (blank).

ENC *char*
Specifies the enclosing character *char*, a character constant of length 1, to be inserted at the start and end of the value depending on the enclosure options **ALL**, **IFNEC** or **STR**.

Parameters ENC and ESC are mutually exclusive. If ENC is specified without *char*, "" (quotation mark) is the default.

Where enclosing characters are to be added to a value, occurrences of the enclosing character that exist as data within the value are duplicated when copied to the target area. This ensures that the CSV value is in a format which is valid for a database or spreadsheet load. e.g. If the enclosing character defined by ENC is "'" (apostrophe), `O'Connell` becomes `'O''Connel''`.

The enclosure options that trigger use of enclosing characters are as follow:

ALL
All values, except null values, will be enclosed.

IFNEC
Values will be enclosed if necessary. i.e. values that contain the delimiter (DLM) character or the enclosing (ENC) character as data. Furthermore, if FILL is assigned a non-blank character, a value containing leading and/or trailing blanks will also be enclosed.

STR
Values will be enclosed if non-numeric. Numeric values must contain at least one numeric digit (0-9) and may only contain leading blank characters, numeric digits and a decimal point. Otherwise, the value is considered to be non-numeric.

Note that non-significant leading zeroes are stripped when a numeric value is copied to the *target* area.

ESC *char*
Specifies the escape character *char*, a character constant of length 1, to be inserted before each occurrence of the following characters which exist as data within the source value:

◊ A delimiter character (as specified by DLM).
◊ An escape character (as specified by ESC).

Parameters ENC and ESC are mutually exclusive. If ESC is specified without *char*, '\' (backslash) is the default.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of COMPRESS for the same source data is unnecessary and so TIMES should never be used on COMPRESS. See common parameter TIMES for details.

CSV Compression Errors:

If an error is encountered during the COMPRESS operation, then SELCOPY return code 8 is set, processing continues and an error message is returned in the SELCOPY list output.

Each compression related error message includes the following element:

Field#*nn*,Len=*nn*	The sequence number and length of the source field in error.
DataLen=*nn*	The length of the value.
TotDestLen=*nn*	The length of the *target* area.

Possible errors relating to compression of data to CSV format are as follow:

CMPRS Error 06: Field#*nn*,Len=*nn* DataLen=*nn* TotDestLen=*nnn* Destn area overflow.
An attempt was made to copy a compressed field value to the target area but the next available position falls outside the target area length (TotDestLen).

CMPRS Error 08: Field#*nn*,Len=*nn* DataLen=*nn* TotDestLen=*nnn* Source contains DLM.
The delimiter character (DLM) was identified within a value in the source area and neither ESC nor ENC parameters were specified. No compression takes place and the value for LRECL is set to 0 (zero).

CMPRS Error 09: Field#*nn*,Len=*nn* DataLen=*nn* TotDestLen=*nnn* Source/Destn overlap.
The source and target areas overlap. No compression takes place and the value for LRECL is set to 0 (zero).

Examples:

Example 1.

The following will compress text defined by a source field of length 80 starting at position 101 of the work area buffer. The target for the compressed data is another field at position 201 of the work area with length defaulting to that of the source field (i.e. 80 bytes). Note that ERROR 615 and RC=44 will be returned if the 80 byte target field is not large enough to contain all the compressed data.

```
COMPRESS   POS 101  LENGTH 80    TO    201
```

Example 2.

Suppose that a source field of length 100 at position 1 of the work area is comprised of 9 character fields of which the first 2 are length 15 and the remaining 7 are length 10. The following example will generate CSV output at position 101 of the work area for the 9 fields using comma (,) delimiters and will enclose all non-null fields in quotation marks (").

The length of the target field is 200 which will easily accomodate the length of the CSV output data. Unused characters following the generated CSV data in the target field will be filled with blanks.

Note that the FLEN parameter specifies only 3 values. The first 2 correspond to the lengths of the first 2 fields, the third to the lengths of the third and subsequent fields.

```
COMPRESS   POS 1, 100    TO    200 AT 101   FLEN=15,15,10   ENC='"'   STR
```

Example 3.

Input text records of fixed length 138, each contain company address data in fixed length fields. In the following example, these records are read into (assigned to) a character declared variable (INREC) which has a defined length of 138. For demonstration purposes only, the INREC value is remapped by declared variables that identify the individual company address fields.

CSV output is generated for the address data using colon (:) delimiters. Furthermore, any occurrence of a colon (:) or slash (/) character in the address data will be prefixed by the default escape character, slash (/). The target field for the CSV data is at position 1 of the work area with a maximum length of 500.

Following execution of the COMPRESS operation the value of internal variable LRECL is automatically set to be the length of the generated CSV data. The WRITE operation writes data to the output data object identified by fname ODD, using the default output field starting at postion 1 of the workarea with length equal to the LRECL value.

```
DECLARE   INREC     CHAR(138)
DECLARE   ID        CHAR(008)    POS INREC+000
DECLARE   CUST      CHAR(030)    POS INREC+008
DECLARE   ADDR1     CHAR(030)    POS INREC+038
DECLARE   ADDR2     CHAR(030)    POS INREC+068
DECLARE   CITY      CHAR(030)    POS INREC+098
DECLARE   ZIP       CHAR(010)    POS INREC+128

OPTION  WORKLEN=1000

READ  IDD   INTO INREC  FILL=' '
COMPRESS         INREC   TO   500 AT 1   FLEN=8,30,30,30,30,10   DLM=':'  ESC
WRITE ODD
```

Example 4.

The following sample SELCOPY output demonstrates how the COMPRESS operation has been used to generate CSV format data from input records containing values in fixed length fields. Each field is terminated by an or symbol (|) which must be translated to a blank before the compression can be successfully performed.

The last FLEN value (27) identifies the length of the 3 fields that occur at the end of each input record (source text). The source text and generated CSV output are printed to illustrate the effect of the COMPRESS operation.

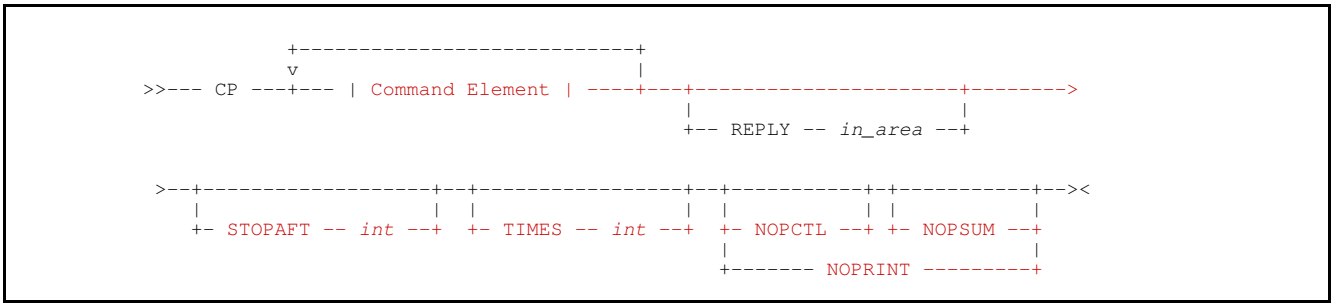
```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)                                                2015/11/11 11:35   PAGE   1
-----------------------------------------------------                                                ----------------   --------


          OPTION      WORKLEN=500   DATAWIDTH=141  PAGEDEPTH=999

     1.   READ  CARD                          INTO    1
     2.   PRINT     TYPE=S  '             '   FROM    SCALE     STOPAFT=1
     3.   PRINT     TYPE=S  'Source Data: '   FROM    1

     4.   TRAN      LRECL AT 1   '|'  ' '                                      * Blank column separators.
     5.   COMPRESS  LRECL AT 1   TO    201    FLEN=5,5,30,10,27   ENC="'"  IFNEC   * Compress as CSV.

     6.   PRINT     TYPE=S  'CSV  Output: '   FROM    201
     7.   SPACE 1

         END


             ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,....0....,....1....,....2....,....3.
Source Data: 10  |2   |Rumour Has It               |Adele    |2011-01-21 08:00:00.000000|2012-08-02 11:30:36.000000|2011-08-19 12:30:11.000000|
CSV  Output: 10,2,Rumour Has It,Adele,2011-01-21 08:00:00.000000,2012-08-02 11:30:36.000000,2011-08-19 12:30:11.000000

Source Data: 10  |3   |Turning Tables             |Adele    |2011-01-21 08:00:00.000000|2012-08-02 11:30:36.000000|                          |
CSV  Output: 10,3,Turning Tables,Adele,2011-01-21 08:00:00.000000,2012-08-02 11:30:36.000000,

Source Data: 10  |4   |Don't You Remember         |Adele    |2011-01-21 08:00:00.000000|2012-08-02 11:30:36.000000|2011-08-19 12:30:14.000000|
CSV  Output: 10,4,'Don''t You Remember',Adele,2011-01-21 08:00:00.000000,2012-08-02 11:30:36.000000,2011-08-19 12:30:14.000000

Source Data: 10  |5   |Set Fire to the Rain       |Adele    |2011-01-21 08:00:00.000000|                          |2011-08-19 12:30:16.000000|
CSV  Output: 10,5,Set Fire to the Rain,Adele,2011-01-21 08:00:00.000000,,2011-08-19 12:30:16.000000


SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL          FSIZE   CI    DSN
 ------      ------      ----     -------  -----          -----   --    ---
    1             4 READ CARD       2048   131 U              4       C:\nbj\ca\sstest48.ctl
    2             1
    3----7        4


    ** SELCOPY/WNT 3.30.002  2015/10/05  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 4. COMPRESS to CSV Format.

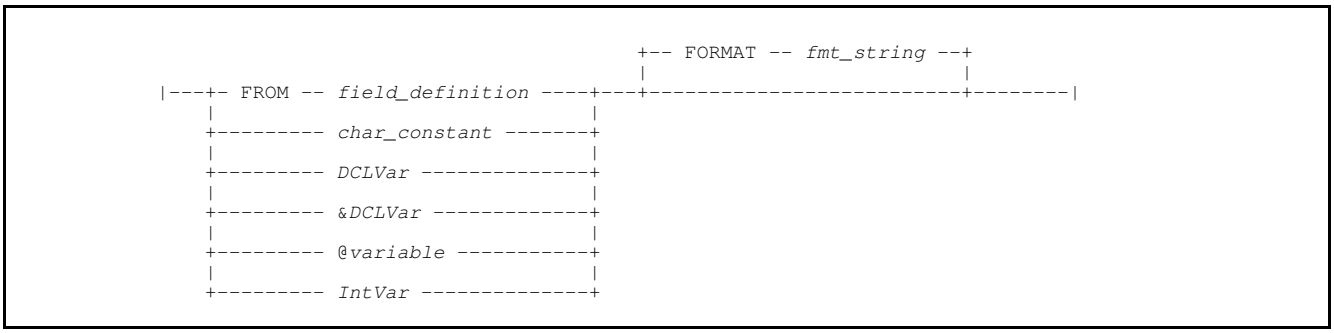
CP

Execute a CP command in a z/VM CMS environment.

Syntax:



Command Element:



Description:

Supported only in z/VM CMS environments, the CP operation will pass a command string to the control program (CP) hypervisor for execution. Selection processing will pause until control is passed back to the SELCOPY program following completion of the command execution.

All text output from the CP command execution is passed into an area specified by the REPLY parameter. If not specified, this area defaults to a field at position 1 (the base address) with length equal to the length of the work area buffer or the LRECL value, if no work area is defined.

The entire command output is terminated with "****" (4 asterisks) within the reply area. If no output is produced by the command, then these terminating asterisks will replace the first 4 characters of the reply area. Individual lines of text within the command output passed into the reply area, are terminated with the NL (new line) character (EBCDIC X'15'). e.g. If only one line of text exists in the output, then a single NL character exists to terminate it.

If the length of the reply area is less than the command output, then the area is filled but the first character is overwritten with the not symbol, "¬" (EBCDIC X'5F'). If the area length is sufficient to contain all the output text but not the additional "****" (asterisks), then the asterisk terminator string is not appended to the output text.

If there is no requirement to process the CP command output in the SELCOPY program or the output is to be directed to the user's terminal, the SYSTEM operation should be used. e.g.

```
SYSTEM  'CP QUERY NAMES'      * Pass the command to the CMS command processor.
```

Execution of a CP command will give return code and condition code values. The return code value set by the CP command execution is assigned to the internal variable RETSYS. If the return code value is 0 (zero) but a non-zero condition code is set, then the RETSYS value is set to 256. The RETSYS value should be tested following each execution of the CP operation in order to establish successful execution.

Beware that CP command strings executed using the CP operation, must be supplied in upper case. This is a limitation of the CP command processor.

The use of the CP operation may have been disabled by the systems programmer using the CBLNAME options variable, **SCPCmds**. If so, an attempt to execute a CP operation will return ERROR 115.

Parameters:

`Command Element`
Specifies an element that constitutes a portion of the command text string.

Multiple command string elements are concatenated, in the order in which they are specified, to construct the complete command string.

Command element values that are of numeric or character numeric data type are automatically converted to decimal character display format. Use &*DCLVar* to obtain the unformatted value of a numeric *DCLVar*.

Each command element may be specified in one of the following formats:

FROM *field_definition*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type.

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL. However, if parameter FORMAT is used to convert the field data to printable hex, specification of a length is mandatory.

Although not necessary, keyword FROM may also be used before a command element specified as a *char_constant* or a *DCLVar* of character data type.

char_constant
A quoted or hexadecimal character constant.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
The name of an @variable that has a non-null value.
If @*variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
The name of a SELCOPY internal variable.

FORMAT *fmt_string*
FMAT
FMT
FORMAT specifies *fmt_string*, a format string used as the character display template for data specified by the command string element.

The source data in a command element of numeric data type is automatically converted to displayable character format (using a CVxC operation). The length of the command element is determined by *fmt_string*, not the length of the command element source.

For command elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For command elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the command element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For a command element specified as an @*variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
`'SS,SSS,SSS,SS9'`	Used for @*variable*, *IntVar* and *DCLVar* representing an integer value.
`'SS,SSS,SSS,SS9.9999'`	Used for *DCLVar* representing a rational value.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

REPLY *in_area*
INTO
REPLY specifies *in_area* which identifies a length and location in storage into which the CP command output will be returned without translation. *in_area* may be specified as a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If *in_area* is a field definition, then, for a type 1 field definition, LRECL may be used as a synonym for LENGTH. If no length is specified, then the default is the length of the work area buffer or the LRECL value, if no work area is defined.

If the length of *in_area* is greater than the length of the command output plus 4 for the "****" terminator string, then the text will be left adjusted within *in_area* and padded with blanks.

The default *in_area* is a field definition starting at position 1 (the base address) with work area or LRECL length.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Examples:

The following sample output is from a SELCOPY program that executes a CP QUERY DASD command and filters the output to display only CP OWNED volumes.

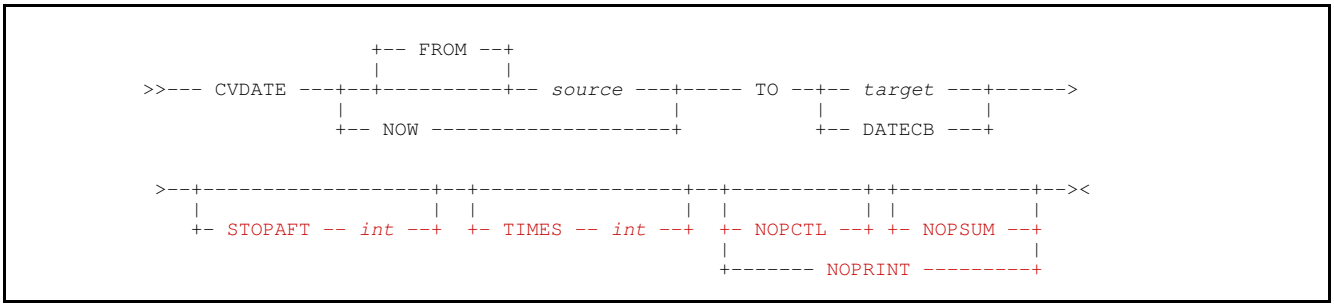
```
SELCOPY/CMS 3.30 at CBL - Bridgend UK (Internal Only)              2016/01/06 17:31   PAGE   1
-----------------------------------------------------              ----------------   --------


          dcl  cpreply    char(4096)

          opt datawidth=60

     1.   cp 'QUERY DASD'  reply cpreply

          if retsys       0
     2.     then plog "No output from CP QUERY DASD"
     3.     then eoj

          if  cpreply     = x'15'  reverse  ptr=@cpend
     4.     then @cplen   = @cpend-cpreply
     5.     else @cpend   = cpreply

     6.   @cpnext         = cpreply

         ==loop==
           ----
          if     @cpnext >= @cpend
     7.     then goto loop_end

          if pos @cpnext, @cpend = x'15'    ptr=@lineterm
     8.     then do   process_line
     9.     then @cpnext  = @lineterm+1
    10.     then goto loop

         ==loop_end==
           --------

    11.   eoj


         ==process_line==
           ------------
          if pos           @cpnext, @lineterm-1  = "OWNED"
    12.     then plog from @cpnext, @lineterm-1

    13.   =return=


    INPUT   SEL SEL                                                          RECORD
    RECNO   TOT ID.          1         2         3         4         5         6  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0  ------
        0     1  12 DASD 0A80 CP OWNED  M01RES   98                                 80
        0     2  12 DASD 0A82 CP OWNED  M01S01   0                                  80
        0     3  12 DASD 0A83 CP OWNED  M01P01   0                                  80
        0     4  12 DASD 0A86 CP OWNED  VMCOM1   11                                 80
        0     5  12 DASD 0A98 CP OWNED  CBLCT1   1                                  80
                   ....,....1....,....2....,....3....,....4....,....5....,....6

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1             1
    2----3        0
    4             1
    5             0
    6             1

                    =loop=
    7             1
    8---10       96

                    =loop_end=
   11             1

                    =process_line=
   12             5
   13            96 =ret=


    ** SELCOPY/CMS 3.30.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                  ** Expiry: 14 Jul 2016 **
```

Figure 5. Execute a CP command.

CVDATE

Convert a date value from one format to another.

Syntax:



Synonyms:

CVDATE	CVDT

Description:

The CVDATE operation converts a specific or current date value to another format and style.

Dates may be converted to and from a number of different date data types using different styles. (e.g. US, European or ISO notation.) See description of date data types and also representation of character dates using a FORMAT string containing date symbols.

Unless NOW is used as the source value specification or DATECB or STYLE=T used as the target, CVDATE does not involve conversion of a time-of-day value.

Parameters:

FROM *source*
FR

References the date value to be converted, *source* may be specified as a date character constant, a declared variable or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

The *source* specification and value must be of a valid date data type. TYPE and STYLE parameters define the source field (character or numeric) data type and date format interpretation respectively.

If specified as a field definition without TYPE, then the data type will default to that of *target*. Otherwise, the default is TYPE=C (character). If STYLE is omitted, the default is STYLE=I (ISO standard date).

TO *target*

References the target to which the converted value will be assigned. The *target* may be a declared variable or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition. If *target* is character data type, then *target* may be also specified as a Type 4 (*field_pFMT*) field definition where the FORMAT string includes date formatting symbols.

The *target* specification includes TYPE and STYLE parameters, identifying the field data type and date format to which the *source* date will be converted.

If specified as a type 1,2 or 3 field definition without TYPE, then the data type will default to that of *source*. If STYLE is omitted, the default is the same as *source*.

NOW

Applicable only as a source value, NOW represents the current date and time. i.e. the date and time at which the CVDATE operation is executed.

DATECB

The DATECB parameter may be used only as a target value. All date fields in the control block referenced by the DATE internal field definition, are refreshed with the date supplied by the source value.

If the source value is NOW, all time fields in the control block are also refreshed, otherwise all time fields are set to zero.

NOPCTL, NOPSUM, NOPRINT

See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of CVDATE for the same source and target values is never necessary and so TIMES should never be used on CVDATE. See common parameter TIMES for details.

Examples:

```
CVDATE   NOW    to DATECB     * Refresh DATE Control Block, as in previous releases.

CVDATE '2007/02/01'          TO  7 AT 131 STYLE=J TYPE=C   * -> '2007032'
CVDATE '2007/02/01'          TO  6 AT 131 STYLE=J TYPE=P   * -> x'0000 2007 032F'
                              * Source still defaults to TYPE=C as it is a date constant.

MOD POS 1 = '2007/06/30'      * Character source.
CVDATE 10 AT 1               TO 15 AT 131 STYLE=J TYPE=U   * Unsigned Packed Decimal target.

                  * Above will fail with RC=8 as no TYPE coded for source, so it
                  * defaults to the same as the destination.

CVDATE 10 AT 1  TYPE=C       TO 15 AT 131 STYLE=J TYPE=U   * Works ok.  Source will
                                                           * still default to STYLE=I.

CVDATE     NOW       TO PREC+40  FORMAT='d Ddd, Mmm ddth yyyy'
                      * Could give:     '7 Sun, Jul  1st 2007'

CVDATE 'June 30th 2007' STYLE=A  TO  DATECB   * Refresh DATE Block, zeroize time fields.

CVDATE 10 AT 1       TO 141  FORMAT='Day d Ddd Mmm ddth ccyy  Dddzzzzzz ddth Mmmzzzzzz yyyy'
                      * Could give: 'Day 6 Sat Jun 30th 2007  Saturday 30th June 2007      '
```

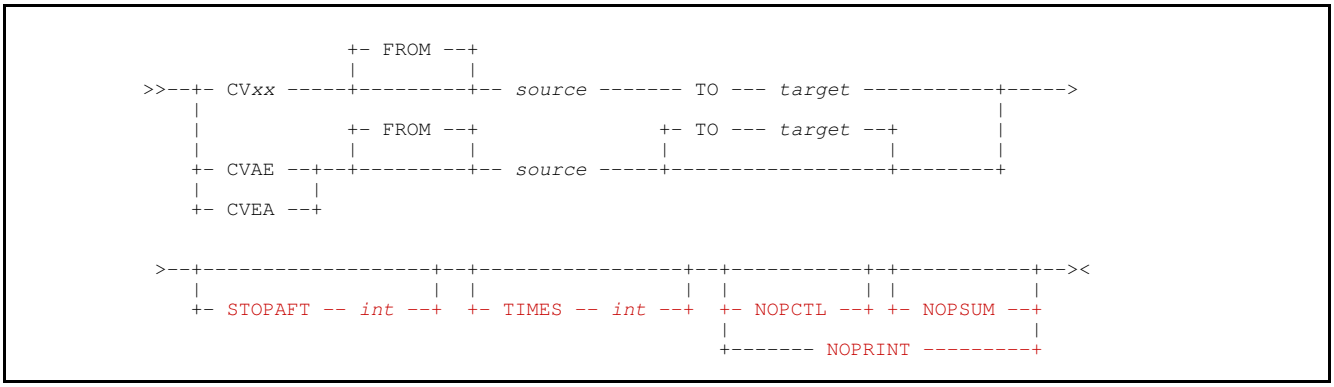
CVXX

CV*xx* converts data from one data type to another where the first *x* represents the source data type and the second *x* the target data type. *x* is a single character symbol representing a data type as follows:

A	ASCII	**F**	Floating Point (Hex or Binary)
B	Binary	**H**	Printable Hex
C	Character	**P**	Packed Decimal
E	EBCDIC	**Z**	Zoned Decimal

Conversion between ASCII and EBCDIC text is documented under the TRAN operation.
(**CVAE** and **CVEA** are synonyms for TRAN EBCDIC and TRAN ASCII respectively.)

Syntax:



Combinations of *xx* are detailed by the following table ("Y" = valid, "." = invalid).

CV*xx*	*x***B**	*x***C**	*x***F**	*x***H**	*x***P**	*x***Z**
B*x*	Y	Y	.	.	Y	Y
C*x*	Y	Y	Y	Y	Y	Y
F*x*	.	Y	Y	.	.	Y
H*x*	.	Y	.	.	.	.
P*x*	Y	Y	.	.	Y	Y
Z*x*	Y	Y	Y	.	Y	Y

Synonyms:

CVAE	CVE
CVBC	UNPKB
CVBP	CVD

CVCB	PACKB
CVCH	CVH
CVCP	PACK

CVEA	CVA
CVPB	CVB
CVPC	UNPK

Description:

A CV*xx* operation will interpret the *source* data as the data type specified by the first *x* character in the operation keyword, convert the data to the data type specified by the second *x* character and assign it to *target*.

Use of data typed field definitions and declared variables diminish the requirement for CV*xx* operations since an assignment statement or MOD operation will perform automatic data type conversion if the *source* and *target* data types are different. e.g. the following operations perform the same action:

```
CVBP   4 AT 11        TO  8 AT 21          * 4-byte binary to 8-byte decimal.
MOD    8 AT 21 TYPE=P  =  4 AT 11  TYPE=B
```

Similarly, use of declared variables, which are always assigned a data type, will be automatically converted by SELCOPY whenever their use requires it. e.g. In an ADD operation where a binary integer value is added to a decimal fixed point value.

Declared variable value assignments also support conversion between data types not supported by CVxx operations. e.g. Conversion of floating point to integer binary or packed decimal.

CVxx operations where the *source* and *target* data types are the same (i.e. CVBB, CVCC CVFF, CVPP and CVZZ) allow reformatting of the value without changing the data type. e.g.

- **CVCC** with FORMAT specified on the *target* field definition can generate a different displayable character format for the same character numeric value.
- **CVFF** with different TYPE F formats (BIN/HEX) and/or lengths on the *source* and *target* specifications, may be used to convert between the different floating point formats.
- **CVBB**, **CVPP** and **CVZZ** may be used to convert the *source* value to *target* having the same numeric value and data type but with a different source data field length.

Field Lengths

The *source* value used and *target* value generated by a CVxx operation are ultimately based on a field in storage, having location, length and data type attributes.

For any of these fields, the maximum length supported by SELCOPY's data conversion depends on the data type as follows:

Data Type	Max Length
Binary	8 Bytes
Character	256 Bytes (1)
Floating Point	8 Bytes (2)
Hexadecimal	512 Bytes (3)
Zoned Decimal	31 Bytes
Packed Decimal	16 Bytes

(1) Maximum of 159 bytes if the *source* of a **CVCH** operation.

(2) Although SELCOPY supports floating point fields of any length up to 8 bytes, it is recommended that fields of length 4 or 8 bytes are used for compatibility with system supported single and double precision fields respectively.

(3) Hexadecimal fields must be an even number of characters.

The length of a *target* value field must be sufficient to store the converted value without truncation or loss of numeric precision. If not, SELCOPY return code 8 is set and the target field updated as follows:

- If *target* is of displayable character or zoned decimal data type, the target field contains "*" (asterisk) characters in every position.
- If *target* is of binary or packed decimal data type, the target value is inserted in the field with truncation of the left most significant bytes.
- If *target* is of floating point or hexadecimal data type, the target value is inserted in the field with truncation of the right most significant bytes.

Parameters:

FROM *source*
References the value to be converted, *source* may be specified as a quoted or hex character constant, a declared variable or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If *source* is a constant or field definition on a CVFx operation, TYPE F BIN/HEX/NAT may be specified to identify the format of the floating point value. In all other cases, a TYPE parameter specified on *source* will be ignored. Instead, the data in the field will be interpreted as being of the source data type implied by the CVxx operation keyword.

If the format of the source data does not comply with the data type used to interpret the data, then SELCOPY return code 8 is set and the conversion is not attempted.

TO *target*
References the target to which the converted value will be assigned. The *target* may be a declared variable or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition. If *target* is displayable character (CVxC) or hexadecimal (CVCH) data type, then *target* may be also specified as a Type 4 (*field_pFMT*) field definition.

If *target* is a constant or field definition on a CVxF operation, TYPE F BIN/HEX/NAT may be specified to identify the format of the floating point value. In all other cases, a TYPE parameter specified on *target* will be ignored. Instead, the data will be converted to the target data type implied by the CVxx operation keyword.

If the length of the target field is invalid or exceeds the maximum for the target data type, then SELCOPY return code 8 is set and the conversion fails.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of CVxx for the same source and target values is never necessary and so TIMES should never be used on CVxx. See common parameter TIMES for details.

CVxB/CVBx - Convert to and from Binary Integer

CVBC, **CVBP** and **CVBZ** operations convert a binary integer value to printable character, packed decimal and zoned decimal formats respectively. **CVBB**, **CVCB**, **CVPB** and **CVZB** operations convert a numeric or numeric character integer value to binary integer format.

A binary integer value may be represented by a field of between 1 and 8 bytes in length and is signed unless the source field is length 1. (i.e. x'FF' is decimal 255, not -1). An 8-byte field may reference a signed binary value between -(2**63) and +(2**63)-1.

When converting to binary integer, if the *source* field contains invalid data or the length of a *target* field is not sufficient to contain the converted value, then SELCOPY return code 8 is set and the *target* value is unchanged.

CVxC/CVCx - Convert to and from Character

CVCH treats a source value as character data, regardless of its assigned data type, and converts each character to printable hex. Each pair of hex digits that constitute a byte, are converted to 2 printable characters. Likewise, **CVHC** will convert printable hex back to its character format. See *"Convert to and from Printable Hex"* below for more detail.

CVCB, **CVCC**, **CVCF**, **CVCP** and **CVCZ** operations convert a numeric character source value to binary integer, displayable character, floating point, packed decimal and zoned decimal respectively.

If the character *source* value is non-numeric, it is treated as having a numeric value of 0 (zero).

CVBC, **CVCC**, **CVFC**, **CVPC** and **CVZC** operations convert a numeric source value to displayable character if the *target* has a defined FORMAT string comprising numeric format symbols. The source value data types are binary, character numeric, floating point, packed decimal or zoned decimal respectively. If no FORMAT string is defined, then the *target* value will be of zoned decimal integer data type. i.e. the CVxC operation is identical to CVxZ.

Any CVxC operation that converts a numeric source to displayable character may be unnecessary if either of the following is true:

1. The source is a declared variable with a default FORMAT defined on its DECLARE operation.
2. The source variable or field definition is to be used as the source of a MOD, LOG, PLOG, PRINT or WRITE operation, and FORMAT is included as part of that source specification to define its displayable character interpretation.

In both of these cases, data type conversion is automatically performed by SELCOPY using temporary work fields in internal storage where necessary. However, if automatic conversion of the same source will be actioned on multiple operations, then use of a CV*xx* or MOD operation to convert the data before hand will make the program execution more efficient.

When converting to displayable character, if the *source* field contains invalid data or the length of a *target* field is not sufficient to contain the converted value, then SELCOPY return code 8 is set and the *target* is filled with "*" (asterisk) characters.

CVxF/CVFx - Convert to and from Floating Point

Floating point values may be of Hexadecimal (HEX) or IEEE-754 binary (BIN) format.

CVFC and **CVFZ** operations convert a floating point value to printable character and zoned decimal formats respectively. **CVCF**, **CVFF** and **CVZF** operations may be used to convert a rational numeric or numeric character value to one of the supported floating point formats.

A floating point value may be represented by a field of between 2 and 8 bytes in length. However, it is recommended that lengths of 4 and 8 should be used to comply with the system standard short and long forms of floating point values respectively.

When specified as a constant or field definition, the format of a *source* or *target* floating point value is determined by a TYPE F and format (BIN/HEX/NAT) specification. e.g.

```
CVCF 20 AT   1             TO 8 AT 101 TYPE=F HEX      * Char to Floating Point.
CVFC  8 AT 101 TYPE=F HEX  TO 1  FORMAT='z,zz9.99999-' * Floating Point to Char.
CVFF  8 AT 101 TYPE=F HEX  TO 8 AT 201 TYPE=F BIN      * IBM to IEEE format.
CVFF  4 AT 101 TYPE=F BIN  TO 8 AT 201 TYPE=F HEX      * IEEE to IBM format.
```

Unless otherwise specified, the format of a *source* or *target* floating point value is determined by the DEFAULTFP environment option. If this option is unset, the default is NATIVE. i.e. the native floating point format for the local system (HEX for z/OS and z/VM CMS, BIN for Windows and Unix systems.)

When converting to floating point, if the length of a *target* field is not sufficient to contain the converted value, then the SELCOPY return code is unchanged and the *target* value is truncated from the right so that low order digits are lost.

For a CVCF operation , a "." (dot/period) in the character source is interpreted as the decimal point and a value of -0 is treated as +0. Similarly, a "." (dot/period) character in the displayable character numeric FORMAT string identifies the location of the decimal point in the *target* of a CVFC operation.

For a CVFC or CVFZ operation, rounding will occur on the fraction portion of the converted rational value if necessary. If converting from a binary floating point value that represents INFINITY or a NAN (SNAN or QNAN), then the literal "inf" or "nan" will be right adjusted in the *target* field, overlaying a formatted 0 (zero) value. Note that sub-normal binary floating point values are unsupported.

On converting to displayable character (CVFC), a maximum of 10 fraction digits are included in the converted value. No limit is imposed on the integer portion of the converted value, other than that specified by the FORMAT string.

```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2015/11/26 14:14   PAGE   1
-----------------------------------------------------              ----------------   --------


          equ irec               1              * Input record.
          equ chr    13 at irec+000 type=c      * Source value 1.
          equ f8h1         irec+017 format='xxxx,xxxx,xxxx,xxxx'
          equ f8b1         irec+041 format='xxxx,xxxx,xxxx,xxxx'
          equ f4b2         irec+065 format='xxxx,xxxx'
          equ f8b2c        irec+084 format='ss,sss,ss9.999,999,9'

          equ f8hex1  8 at      201 type=f hex    * Field as IBM  Base 16, len 8 (long).
          equ f8bin1  8 at      211 type=f bin    * Field as IEEE Base  2, len 8 (long).
          equ f4bin2  4 at      221 type=f bin    * Field as IEEE Base  2, len 4 (short).

          option   worklen=300  datawidth=105

     1.   read card       into irec       fill

          if pos irec  = '.'
          or pos irec  = 'C'
          or pos irec  = '-'
     2.    then print   from  irec  len=105   type=s
     3.    then goto get


     4.   cvcf  chr     to  f8hex1
     5.                             cvch f8hex1  to  f8h1
     6.   cvff  f8hex1  to  f8bin1
     7.                             cvch f8bin1  to  f8b1
     8.   cvff  f8hex1  to  f4bin2
     9.                             cvch f4bin2  to  f4b2
    10.   cvfc  f8bin1  to  f8b2c

    11.   print         from  irec  len=105   type=s

         end  *       f8h1                    f8b1                   f4b2                f8b2c


....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10....+
Char Source   |Src->FltHex(1)         |FltHex(1)->FltBin(1)   |FltHex(1)->FltBin(2)|FltBin(1)->Char     |
--------------|-----------------------|-----------------------|--------------------|--------------------|
   10         |X'41A0,0000,0000,0000' |X'4024,0000,0000,0000 '|X'4120,0000'        |       +10.000,000,0|
1.2           |X'4113,3333,3333,3333' |X'3FF3,3333,3333,3333 '|X'3F99,9999'        |        +1.200,000,0|
     -1.2     |X'C113,3333,3333,3333' |X'BFF3,3333,3333,3333 '|X'BF99,9999'        |        -1.200,000,0|
240           |X'42F0,0000,0000,0000' |X'406E,0000,0000,0000 '|X'4370,0000'        |      +240.000,000,0|
36.584        |X'4224,9581,0624,DD30' |X'4042,4AC0,8312,6E98 '|X'4212,5604'        |       +36.584,000,0|
0.0000052     |X'3C57,3DD4,76E5,39AC' |X'3ED5,CF75,1DB9,4E6B '|X'36AE,7BA8'        |        +0.000,005,2|
63904.0089    |X'44F9,A002,4745,38F0' |X'40EF,3400,48E8,A71E '|X'4779,A002'        |   +63,904.008,900,0|
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+...10....+

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1            11 READ CARD        2048   105 U             11       Z:\cd\sman\1340\SMXCVFF.CTL
    2----3        4
    4---11        7


    ** SELCOPY/WNT 3.30.002  2015/10/05  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 6. CVCF, CVFF, CVFC and CVCH Examples.

CVCH/CVHC - Convert to and from Printable Hex

CVCH converts each byte in the *source* to displayable hex format. i.e. each pair of hex digits that represent a single character byte is converted to 2 consecutive, displayable characters in the *target* value.

The hex digit characters are represented by the code points for characters 0 to F in the local code page. These invariant code points are [x'F0'-x'F9',x'C1'-x'C6'] in EBCDIC and [x'30'-x'39', x'41'-x'46'] in ASCII. e.g. In z/OS and z/VM CMS, the *source* value `SEL1`, which has EBCDIC code points `x'E2C5,D3F1'`, will convert to displayable hex characters `E2C5D3F1`.

The CVCH *target* field supports specification of a FORMAT string comprising printable hex format symbols. e.g.

```
CVCH 'abcd' ASCII     TO 1 FORMAT="  'xx,xx xx,xx' = ASCII code.  "
CVCH 'abcd' EBCDIC    TO 1 FORMAT="  'xx,xx xx,xx' = EBCDIC code. "
```

Unless FORMAT is specified, the length of the CVCH *target* value is twice the length of the *source* value.

CVHC converts pairs of hex digits in displayable character format to a single byte hex value. The hex value references a code point which may be a displayable character in the code page of the local terminal. e.g. In Windows and Linux, the *source* displayable character value `414263` will convert to hex digits `x'414263'` which reference the 3 displayable ASCII code points for characters `ABc`.

The length of the CVHC *target* value is half the number hex digits in the *source* value. The following conditions apply to CVCH operations:

1. Occurrences of "," (comma) and blank characters in the displayable hex character *source* value are ignored. e.g. `5,30C,4Cbbb43` will convert to `x'530C4C43'`.
2. Apart from "," (comma) and blank characters, any displayable character in the *source* value that is not a valid hexadecimal digit, will set SELCOPY return code 8 and fill *target* with "*" (asterisk) characters. Note that hex digit characters may be represented as upper or lower case printable characters.
3. If the *source* value contains an odd number of hex digit characters, then the last hex digit will convert to printable character "*" (asterisk) in *target*. e.g. In an ASCII based system `41,42,4` will convert to printable characters `AB*`.

CVxP/CVPx - Convert to and from Packed Decimal Integer

CVPB, **CVPC** and **CVPZ** operations convert a packed decimal integer value to binary integer, printable character and zoned decimal formats respectively. **CVBP**, **CVCP**, **CVPP** and **CVZP** operations convert a numeric or numeric character integer value to packed decimal integer format.

A signed packed decimal integer value may be represented by a field of between 1 and 16 bytes in length allowing values of up to 31 decimal digits.

When converting to packed decimal, if the length of a *target* field is not sufficient to contain the converted value, then SELCOPY return code 8 is set and the *target* value is truncated from the left so that the high order digits are lost.

```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2015/11/26 11:23   PAGE   1
-----------------------------------------------------              -----------------   --------



     1.  READ CARD   WORKLEN 1000   FILL

     2.  CVCP 10 AT  1   TO  4 AT 22     * PACK (4th record is truncated)
     3.  CVPB  4 AT 22   TO  3 AT 32     * CVB
     4.  CVBP  3 AT 32   TO  4 AT 40     * CVD
     5.  CVPB  4 AT 22   TO  2 AT 52     * CVB  (1st 4 records are truncated)
     6.  CVPZ  4 AT 22   TO  7 AT 58     *

     7.  CVCH  4 AT 22   TO       22     * Convert to hex notation.
     8.  CVCH  3 AT 32   TO       32     * Note that destination overwrites source.
     9.  CVCH  4 AT 40   TO       40     *
    10.  CVCH  2 AT 52   TO       52     *

    11.  PRINT 'Char(10)   Comments   Pack(4)  Bin(4)   Pack(4)  Bin(2)    Zoned' STOPAFT=1
    12.  PRINT '---------  --------  --------  ------  --------  ------    -----' STOPAFT=1
    13.  PRINT LENGTH 80                * Print the card area.

         END


    INPUT   SEL SEL                                                                                  1  RECORD
    RECNO   TOT ID.          1         2         3         4         5         6         7         8         9         0  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
        1     1  11 Char(10)   Comments   Pack(4)  Bin(4)   Pack(4)  Bin(2)    Zoned                                        9
        1     1  12 ---------  --------  --------  ------  --------  ------    -----                                        9
        1     1  13 10,460.98            1046098C  0FF652  1046098C    F652  1046098                                      9
        2     2  13 4.240,22             0424022C  067856  0424022C    7856  0424022                                      8
        3     3  13 3 186 40             0318640C  04DCB0  0318640C    DCB0  0318640                                      8
        4     4  13 123456789  Truncate  3456789C  34BF15  3456789C    BF15  3456789                                     19
        5     5  13 -176       Negative  0000176D  FFFF50  0000176D    FF50  000017O                                     19
        6     6  13 - 2 20     Negative  0000220D  FFFF24  0000220D    FF24  000022}                                     19
        7     7  13 25 CR      Negative  0000025D  FFFFE7  0000025D    FFE7  000002N                                     19
        8     8  13 20 f 3gk   Negative  0000203D  FFFF35  0000203D    FF35  000020L                                     19
        9     9  13 cr 20      Positive  0000020C  000014  0000020C    0014  0000020                                      19
       10    10  13 2 CR 0     Positive  0000020C  000014  0000020C    0014  0000020                                      19
       11    11  13 2 #(=0     Positive  0000020C  000014  0000020C    0014  0000020                                      19
                 ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,....0

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1            11 READ CARD       2048     19 U            11       Z:\cd\sman\1340\SMXCVPB.CTL
    2            11      (***01 RETCD=8***)
    3----4       11
    5            11      (***04 RETCD=8***)
    6---10       11
   11---12        1
   13            11

***WARNING*** (SEL----5)        8 = RETURN CODE FROM SELCOPY

    ** SELCOPY/WNT 3.30.002  2015/10/05  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 7. CVCP, CVPB, CVBP, CVPZ and CVCH Examples.

CVxZ/CVZx - Convert to and from Zoned Decimal Integer

CVZB, **CVZC**, **CVZF** and **CVZP** operations convert a zoned decimal integer value to binary integer, printable character, floating point and packed decimal formats respectively. **CVBZ**, **CVCZ**, **CVFZ**, **CVPZ** and **CVZZ** operations convert a numeric or numeric character value to packed decimal integer format. Note that a CVxC operation, where no FORMAT string is specified for *target*, is equivalent to a CVxZ operation.

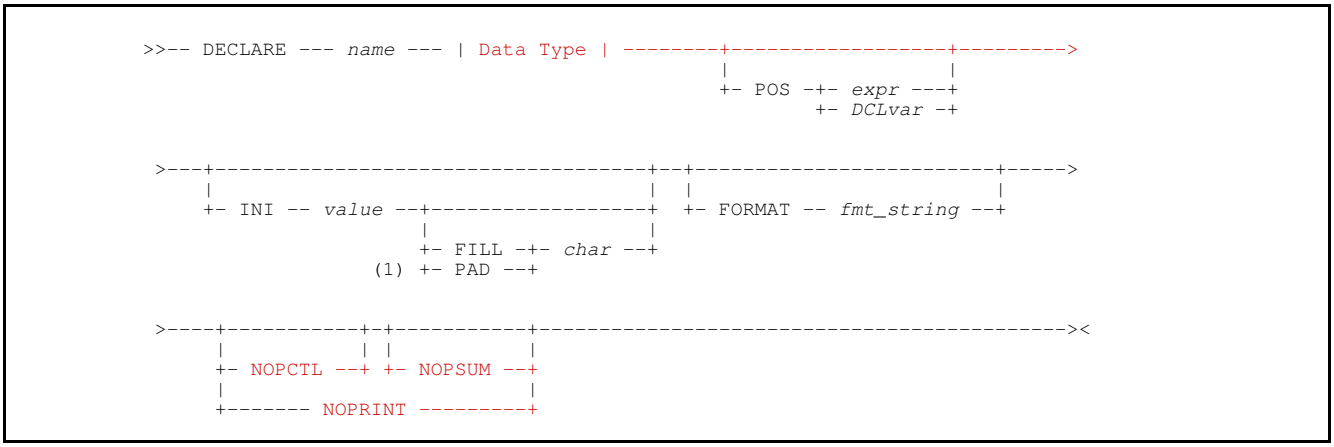
A signed zoned decimal integer value may be represented by a field of between 1 and 31 bytes in length allowing values of up to 31 decimal digits.

When converting to zoned decimal, if the *source* field contains invalid data or the length of a *target* field is not sufficient to contain the converted value, then SELCOPY return code 8 is set and the *target* is filled with "*" (asterisk) characters.

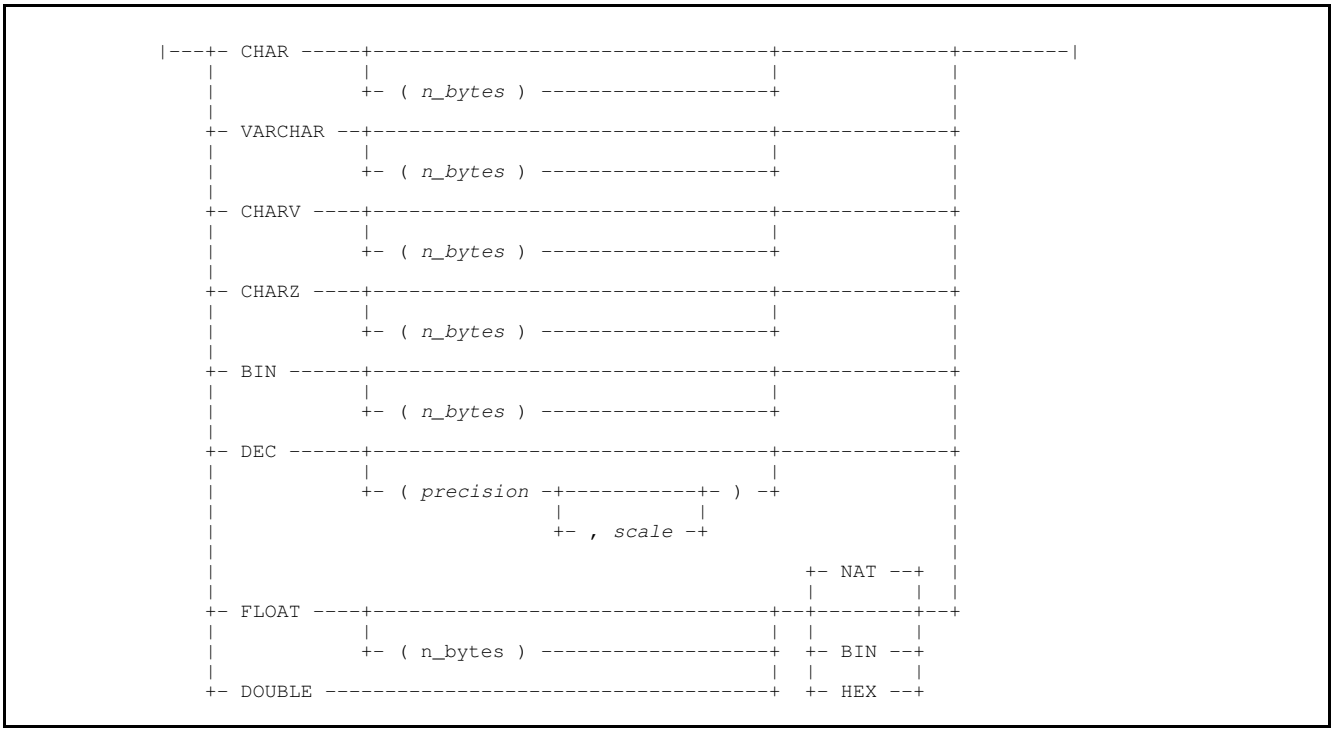
DECLARE

Declare a variable and optionally assign a non-default initial value.

Syntax:



Data Type:



Syntax Notes:

(1) Parameter keyword FILL is applicable to fixed length character data types only.

Synonyms:

DECLARE	DCL

Description:

The DECLARE operation defines a declared variable which may be referenced, in place of its assigned value, throughout the program's control statements. It is classed as one of the control statement analysis operations.

The value assigned to the declared variable is held in a source field defined either within its own area of dynamic storage or as an overlay of a predefined area of storage (e.g. the user work area). Like any field definition, the source field has position, length and data type.

The contents of the source field belonging to a declared variable of any data type may be referenced by prefixing the variable name (*name*) with "&" (ampersand). i.e. *&name* is the unformatted (character) value of the declared variable with length equal to the length of the source field.

However, *&name* may also be specified with an offset (arithmetic expression) to identify the start position of a field definition relative to the declared variable's source field. In this case, *&name* references a location in storage only and the length of the variable source field is no longer implied. e.g.

```
DECLARE       COUNT   DEC(7)   INI=256   FMT='Z,ZZ9'

PRINT         COUNT                      * Prints: '  256'
PRINT        &COUNT              FMT='XXXX,'    * Prints: '0000,256C'
PRINT  FROM  &COUNT+1  LENGTH=3   FMT='XX,'      * Prints: '00,25,6C'
```

If required, the source field may overlay an area of storage within the source field of another declared variable or within the allocated work area (see parameter POS). However, if all data is processed as declared variable values without overlaying work area storage, allocation of a work area is unnecessary.

The value assigned to a declared variable will be stored in the source field in the defined data type format. This value may be updated by assignment statements and operations executed during selection time processing.

Unless the INI parameter is specified on the DECLARE operation, the variable is assigned an initial value of blank for a character data type, or 0 (zero) for a numeric data type.

Parameters:

name
The programmer defined name used to reference the declared variable within control statements.

Data Type
The data type assigned to the variable source field.

CHAr (*n_bytes*)
C
A character fixed length value of length specified by the parenthesised integer value, (*n_bytes*).

Assignment of a value that has a length less than *n_bytes* will be padded with blank characters.

If (*n_bytes*) is omitted then the default length is determined as follows:

1. If parameter INI is specified, default is the length of the initial *value*.
2. If parameter FORMAT is specified, default is the width of *fmt_string*.
3. Default is 1.

VARCHAR (*n_bytes*)
Vchar
A character variable length value of maximum data length specified by the parenthesised integer value, (*n_bytes*). The length of the character data is maintained in a 2-byte binary length field prefix to the character data. Any reference to a VARCHAR variable does not include this length field.

Following assignment of a value, any unused characters following the value text in the source field are unchanged. The 2-byte length field prefix is updated with the length of the assigned value.

If (*n_bytes*) is omitted, the default length is 124.

CHARV (*n_bytes*)
CHV
CHARVARYING
A character varying length value of maximum data length specified by the parenthesised integer value, (*n_bytes*).

Following assignment of a value, any unused characters following the value text in the source field are padded with blanks up to length *n_bytes*. The 2-byte length field prefix is updated with the length of the assigned value.

If (*n_bytes*) is omitted, the default length is 124.

CHARZ (*n_bytes*)
CHZ
CHAZ
NTS
Zchar
CSTRING
CSTR

A character variable null terminated value with a maximum data length of 1 less than the parenthesised integer value, (*n_bytes*). i.e. the *n_bytes* length includes the null terminator character (x'00').

Following assignment of a value, the null terminator is appended to the value in the source field and any unused characters following the null terminator are unchanged. If INI is not specified, a null value is assigned.

If (*n_bytes*) is omitted, the default length is 126.

BIN (*n_bytes*)
B

A binary integer value represented by a field of length between 1 and 8 bytes as specified by the parenthesised integer value, (*n_bytes*).

If (*n_bytes*) is omitted, the default is 4.

DEC (*precision*,*scale*)
D

A decimal integer value with a maximum precision of 31 as specified by the parenthesised integer value, (*precision*). If a second integer value, *scale*, is specified within the parentheses following *precision*, the variable represents a decimal fixed point value.

A *scale* has a minimum and default value of 0 and maximum value of *precision*.

If (*precision*,*scale*) is omitted, the default is (5,0).

FLOAT (*n_bytes*)
FLT
FP
F

A floating point value represented by a field of length between 1 and 8 bytes as specified by the parenthesised integer value, (*n_bytes*).

The format of the floating point value is determined by one of the following sub-parameters.

BIN (BFP)	IEEE-754 Base 2 binary floating point.
HEX (HFP)	IBM Base 16 hexadecimal floating point.
NAT (NATIVE)	Floating point format native to the local machine architecture.

If (*n_bytes*) is omitted, the default is 4. The default floating point format is that defined by environment option DEFAULTFP.

DOUBLE
DBL

A double precision floating point value. DOUBLE is a synonym for FLOAT(8).

FORMAT *fmt_string*
FMAT
FMT

Specifies a numeric format string that defines a numeric value interpretation for character variables and a character display format for numeric variables.

Character Data Types:

FORMAT is not applicable to variables of data type VCHAR, CHARV or CHARZ. However, for a variable of fixed character data type (CHAR), *fmt_string* defines the value as numeric character data.

The numeric character value has a precision defined by the sum of all digit, zero suppression and floating sign control symbols within *fmt_string*. Furthermore, if a decimal point control symbol exists, the numeric value also has a scale which indicates a fraction portion for the value.

The *fmt_string* width must match the length of the variable definition, otherwise ERROR 205 is returned. If, however, both the INI parameter and *n_bytes* value have not been specified, the default length of the variable is defined as the width of *fmt_string*.

Numeric Data Types:

For numeric data types, *fmt_string* defines the default character format applied when the variable is used in an operation to reference a character source value. (e.g. on a PRINT, WRITE or MOD operation).

If *fmt_string* does not include digit control characters that represent all fraction digits of a DEC or FLOAT value, then rounding will occur to the lowest decimal place represented by *fmt_string*.

If no FORMAT parameter is specified for a numeric variable, the following defaults are used:

fmt_string	Numeric Data Type
'SS,SSS,SSS,SS9'	Variables supporting integer values: BIN and DEC(p).
'SS,SSS,SSS,SS9.9999'	Variables supporting rational values: DEC(p,s), FLOAT, DOUBLE.

INI *value*
INIT

Specifies *value*, an initial value to which the variable will be assigned.

For character data types, *value* must be specified as a quoted or hex character constant with or without ASCII/EBCDIC encoding keywords. The default initial value for character variables is blank with blank padding for fixed length character data type.

For numeric data types, *value* is specified as a numeric or numeric character constant. The default initial value for numeric variables is 0 (zero).

FILL *char*
PAD

Applicable only to variables of fixed length character data type, FILL defines the pad character (*char*) to be used when the length of the value specified by INI is less than the declared variable length.

The pad *char* may be specified as a quoted or hex character constant of length 1.

The default is the blank character.

NOPCTL, NOPSUM, NOPRINT

See common parameters NOPCTL, NOPSUM, NOPRINT for details.

POS *expr | DCLvar*
P

Specifies a location in storage at which the variable's source field is defined.

Dynamic storage is not allocated for the variable's source field as is default. Instead the POS location references an area of dynamic storage that has already been allocated for data input (input buffer or work area) or another declared variable.

Using this method, a variable may be declared so that its value is updated as the area containing its source field is updated. e.g. A variable with a source field defined within the area of storage into which records are read.

If the POS location is specified as an arithmetic expression, *expr*, or a declared variable, *DCLvar*, of numeric or numeric character data type, then the value represents a position from the base storage address.

If the POS location is a declared variable, *DCLvar*, of character data type or an offset from that *DCLvar* (e.g. *DCLvar*-2), then it represents a position from the dynamic storage location at which the *DCLvar* source field is defined.

Examples:

```
  DCL   irec   CHA  (100)              * Field in dynamic storage.  (No POS.)
  DCL   rec2   cha  ( 50)              * Field in dynamic storage.  (No POS.)
  DCL   ABC    Char  (10)   POS 1001   * Field at POS 1001 in WORKLEN area.
  DCL   totc   CHAR FMAT='zz,zz9.99'   * Field length omitted.

  dcl   tot1   DEC                     * Default: (5,0) needing 3 bytes.
  dcl   tot2   dec   (5)               * Same as (5,0) needing 3 bytes.
  dcl   tot3   dec   (9,3)             * 5 bytes, with 3 places of decimal.
  dcl   tot4   d     (7,2)             * 4 bytes, with 2 places of decimal.
  dcl   tot5   d    ( 31, 6 )          * 16 bytes (max) with 6 places of decimal.

  dcl   totb1  bin                     * Default: (4) needing 4 bytes.
  dcl   totb2  bin   (6)               * 6-byte binary field.
  dcl   totb3  bin   (8)               * 8-byte (max) binary field.

  dcl   totf1  float                   * Default: (4) needing 4 bytes.
  dcl   totf2  flt   (4)               * Same as default FLOAT.
  dcl   totf3  double                  * Default: (8) needing 8 bytes.
  dcl   totf4  float (8)               * Same as default DOUBLE.
  dcl   totf5  double(4)               * Same as default FLOAT.

  dcl   fld1   DEC   (19,6)            * Field in dynamic storage. (10 bytes.)
  DCL   fld2   CHA   (90)              * Field in dynamic storage.
  dcl   irec   char  (100)  pos=fld1   * Field overlaying fld1 and fld2.
```

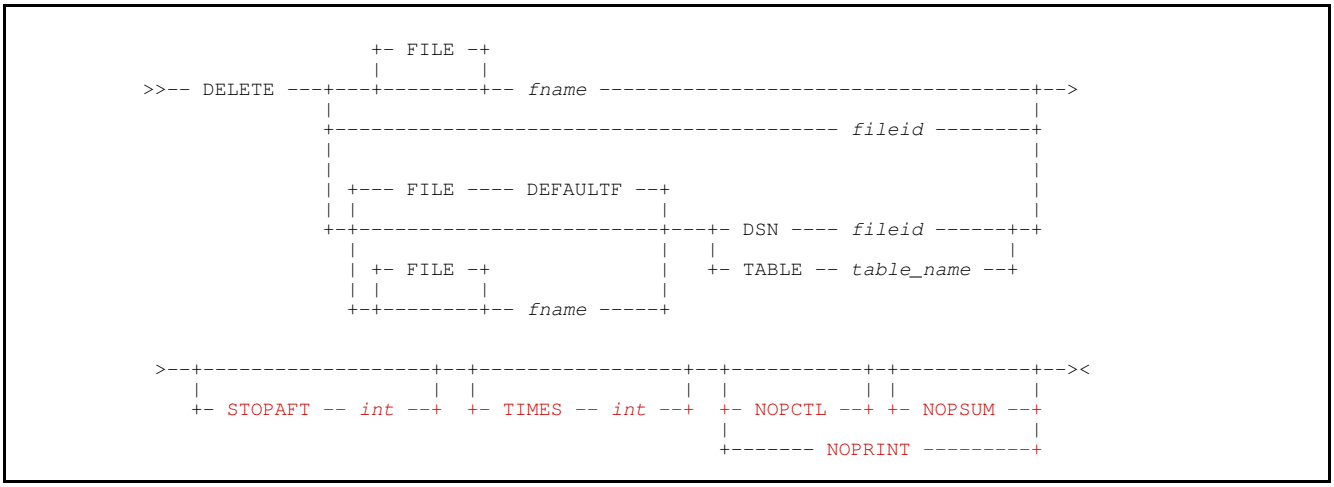
The following control statements demonstrate use of the above declared variables in expressions, in assignment statements and as the source or target of an operation.

```
read abcfil  into irec              * ERROR 546 if input rec exceeds size of DCL var, irec.
                                    * Prev rec residue not cleared unless FILL coded.
irec = 'ABC'                        * Pos 4 up to end of irec will be blank padded.
irec+20 = 'ABC'                     * Pos 1 to 20 of irec will be unchanged.
                                    * Pos 21 to 23 of irec will be set to 'ABC'.
                                    * Pos 24 up to end of irec will be padded with
                                    * the FILL char, which by default is blank.

rec2 = irec                         * Contents of irec will be copied to irec, truncated to
                                    * fit rec2, with no error given.
irec = rec2                         * Contents of rec2 will be copied to rec2 and the
                                    * remainder of irec padded with the FILL char.

if irec = rec2                      * The shorter field is padded with the FILL char.
  then tot5 = totf4                 * Arithmetic assignment follows normal conversion
                                    * rules.  RC=8 is possible if too large for destination.

if tot5 = totf4                     * Arith comparison follows normal conversion rules.
  then @xyz = irec                  * Sets @xyz to point at the 1st byte of irec.
  then @xyz = irec+20               * Sets @xyz to point at pos 21 of irec.

@abc = tot1                         * Sets @abc to the VALUE held in the arithmetic
                                    * field, tot1.  It does NOT point to it.
@def = &tot1                        * Sets @def to point to the source field of tot1.

add tot1 to totb1   into totf5      * Sets totf5 to the value tot1 + totb1.
totf5 = tot1 + totb1                * Does the same thing.
add tot1 to totb1   into @tot       * Sets @tot  to the value tot1 + totb1.
@tot = tot1 + totb1                 * Does the same thing.
                                    * RC=8 is possible if too large for an @ ptr value
                                    * which is held as a 4 byte binary number.

if irec = 'x'  ptr=@x               * Scans irec for 'x' and sets the pointer @x to
                                    * the 1st 'x' found. If no 'x' found, @x is set to NULL.
   * Not suitable for arithmetic DCL vars, but if really necessary, use  POS=refname on a DECLARE
   * operation for a CHA var overlaying the same storage, or use &DCLvar to refer to its source field.
```

DELETE

Delete a record or table row read from an input data object source.

Syntax:



Synonyms:

DELETE	DEL DLET

Description:

Delete of current input record is supported for z/OS VSAM and z/VM CMS VSAM data sets with organisation KSDS or RRDS, ODBC input tables and Micro Focus relative and indexed files.

For Micro Focus files and VSAM data sets, input parameter UPD must have been specified for the data object in order that it is opened for update. If not, ERROR 26 will be returned indicating an input/output conflict on the same file name. UPD may be specified on the READ or OPEN operation.

Before a record or table row can be deleted, a successful READ operation must first be executed in order to establish an input position within the file or table. The DELETE operation will delete the record or table row which was last read from the specified input object. The deleted record or table row may have been read into any position of a defined work area or a declared character variable.

Any attempt to delete a record or table row following a failed READ for the input object will return an error and then terminate the SELCOPY run.

Following execution of DELETE, sequential processing of the input object will continue at the next record or table row following the deleted record or row.

UPDATE operations, DELETE operations and, for VSAM and Micro Focus input only, INSERT operations may exist for the same data object within a SELCOPY program.

DELETE is not supported for input read via a CAT sub-operation.

ODBC Table Delete

DELETE performs the positioned form of an SQL DELETE statement on the current row and can only delete rows belonging to the first (or only) source table referenced by the SQL query FROM clause. The searched form of SQL DELETE may be actioned using the ODBC operation.

```
READ  EMP   SQL='SELECT  WORKDEPT    FROM DSN8910.EMP'

IF WORKDEPT = 'E11'
  THEN DELETE  EMP       * Delete when WORKDEPT='E11'
```

Parameters:

fileid
For file object delete where no specific *fname* has been defined on the READ operation, *fileid* may be used to identify the same file referenced on READ. *fileid* is a fileid clause specifying the name by which the input file is known to the local system.

If *fname* is specified and is already associated with a *fileid*, then re-specification of *fileid* on the DELETE operation is unnecessary.

If not specified as the DSN parameter value, then specification of FILE is invalid and the associated *fname* is derived from *fileid* as described by *fileid* for the READ operation.

fileid may be specified as an unquoted literal, a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If specified as a variable or field definition, *fielid* must be an argument to parameter DSN. Also, *fileid* specified on the READ operation for the input file, must also be a variable or field definition (i.e. a dynamic value).

DSN *fileid*
Specifies the name of the input file. See *fileid* for details.

If no *fname* is specified on a DELETE operation which uses DSN, then an *fname* of DEFAULTF is used by default.

FILE *fname*
F
Identifies the file name assigned to the input data object from which a record or row will be deleted.

fname must match the specified (or derived) file name on the READ operation for the object from which records will be deleted. It may only be specified as an unquoted literal.

The *fname* value may be specified with or without the FILE parameter keyword.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TABLE *table_name*
TAB
For ODBC table object delete where no specific *fname* has been defined on the READ operation, TABLE *table_name* may be used to identify the same base table referenced by the READ TABLE parameter.

If *fname* is specified and is already associated with a *table_name*, then re-specification of *table_name* on the DELETE operation is unnecessary.

If no *fname* is specified on a DELETE operation which uses TABLE, then an *fname* of DEFAULTF is used by default.

The *table_name* value may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

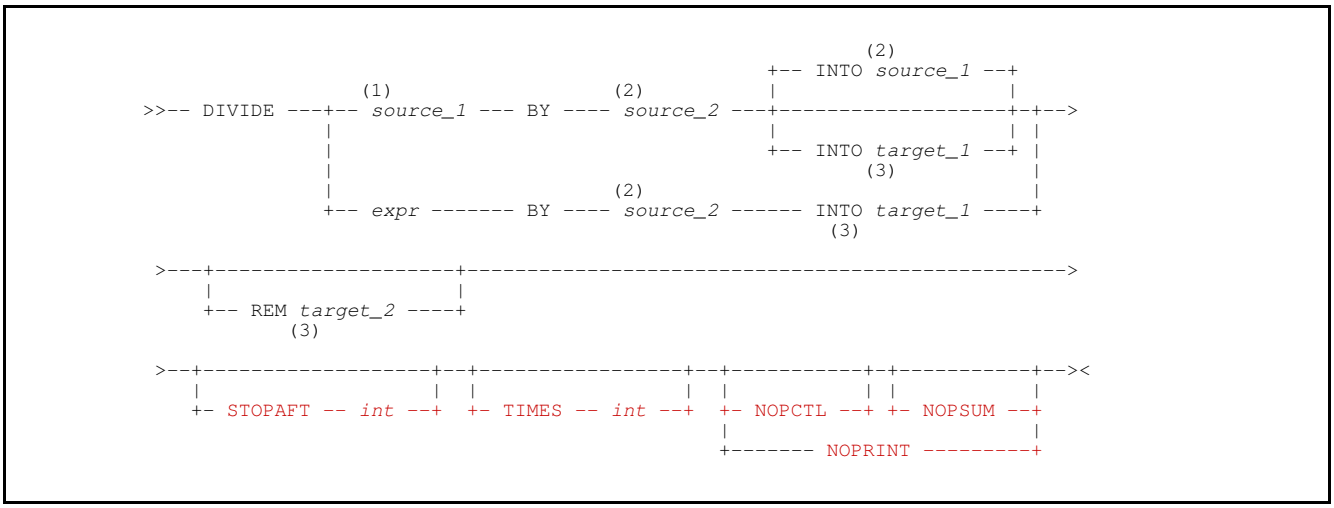
If *table_name* is specified as a variable or field definition, then *table_name* specified on the READ operation for the same input table, must also be a variable or field definition (i.e. a dynamic value).

TIMES *int*
Although supported as a parameter on DELETE, TIMES must never be used since a second attempt to delete the same record or table row will always fail, resulting in ERR538. See common parameter TIMES for details.

DIVIDE

Performs an arithmetic division of one, signed numeric source value by another. The resultant quotient value, and optionally the remainder value, are assigned to target variables or stored in target fields.

Syntax:



Syntax Notes:

(1)	***source_1***	must be of numeric or character numeric data type and may be specified as a declared variable or as a field definition of type 2 (*field_p1p2*) or type 3 (*field_nATp*). If INTO *target* is specified, *source_1* may also be specified as a constant value or an arithmetic expression (*expr*).
(2)	***source_2***	must be of numeric or character numeric data type and may be specified as a constant, an arithmetic expression (*expr*), a declared variable or as a field definition of type 2 (*field_p1p2*) or type 3 (*field_nATp*).
(3)	***target_1*** ***target_2***	may be specified as a declared variable or as a field definition of type 2 (*field_p1p2*), type 3 (*field_nATp*) or type 4 (*field_pFMT*).

Synonyms:

DIVIDE	DIV

Description:

Variables and field definitions that constitute the source values and target of the DIV operation may be of different numeric or character numeric data types.

When the source and target values are of different data types and/or if an arithmetic expression or constant value is specified as a source field, then values are converted to a common numeric data type prior to performing the division. The selected common data type will have a precision and scale which is sufficient to manage the maximum precision and scale of the resultant value.

Any rational number having a fraction value, will be stored internally as a double precision (8-byte) floating point number. Therefore, all fields in the operation will be converted to double precision floating point data type and then floating point arithmetic used. If this is the case, use of the REM parameter to specify the target of a remainder value is invalid and will return ERROR 192.

If a target or source value is expressed as a field definition without a TYPE parameter, the default data type attributed to that field will be the same as the first field definition within the command syntax to have TYPE specified. If no such field definition exists, the default data type is packed decimal integer (TYPE=P). e.g.

```
DIV  4 AT 1 TYPE=C  BY  4 AT 6 TYPE=B  INTO 4 AT 11          * Default TYPE=C
DIV  4 AT 1 TYPE=B  BY  4 AT 6 TYPE=C  INTO 4 AT 11          * Default TYPE=B
DIV  4 AT 1         BY  4 AT 6         INTO 4 AT 11 TYPE=C   * Default TYPE=C
DIV  4 AT 1         BY  4 AT 6         INTO 4 AT 11          * Default TYPE=P
```

If the division involves rational numbers, rounding will occur as required on the fractional part of the quotient value based on the number of fractional digits (scale) of the *target_1* definition. e.g. If the target variable or field has no scale, the quotient value will be rounded to the nearest integer value by adding 0.5 and dropping the fractional portion.

Parameters:

source_1
Represents a rational numeric value (dividend) to be divided.

If no *target* is specified, then *source_1* is also the target of the operation to which the resultant value (quotient) will be assigned. If this is the case, *source_1* must be a declared variable or a field definition.

BY *source_2*
Represents a rational numeric value (divisor) by which the value specified by *source_1* will be divided.

expr
Represents a numeric or character numeric constant, or an arithmetic expression to be used as the dividend value.

Note that the divisor (*source_2*) may also be specified in this way. The distinction is made for *expr* and *source_1* because *source_1* must also be a valid target if INTO is not specified. Therefore, if *expr* is used, INTO *target_1* is mandatory.

INTO *target_1*
Identifies the declared variable or field definition that is the target of the operation and to which the resultant value (quotient) is assigned. If the first source value is *expr*, then a *target_1* is mandatory.

If required, data conversion and decimal rounding will be performed on the resultant value before it is assigned to *target_1*.

Default is *source_1*.

REM *target_2*
Valid only if *source_1*, *source_2* and *target_1* are defined with data types that can only represent integer values. REM *target_2* identifies the declared variable or field definition that is the target for the remainder value.

The remainder is the integer value remaining following a divide operation performed on integer values.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Examples:

```
  DECLARE  FVAL     FLOAT BIN                  * Floating Point.
  DECLARE  BVAL     BIN                        * Binary integer.
  DECLARE  PVAL     DEC(6)                     * Decimal No scale.

  EQU      OREC    1001                        * Output record area.

  OPTION   WORKLEN 2000                        * Work area buffer.

  @METRES     =    3 AT 1 TYPE=P               * Assign @variable value.

  DIVIDE  20               BY     6             INTO FVAL
  DIVIDE  20               BY     6             INTO BVAL   REM PVAL
  DIVIDE  4 AT 11 TYPE=B   BY  8 AT 23 TYPE=P   INTO PVAL   REM 10 AT 81 TYPE=Z
  DIVIDE  @METRES          BY  1000             INTO 2 AT 53 TYPE=P
  DIVIDE  16.431           BY    20             INTO 8 AT 61 TYPE=F BIN
  DIVIDE  FVAL             BY  8 AT 61 TYPE=F   INTO  OREC+11   FMT='S,SS9.99'
```

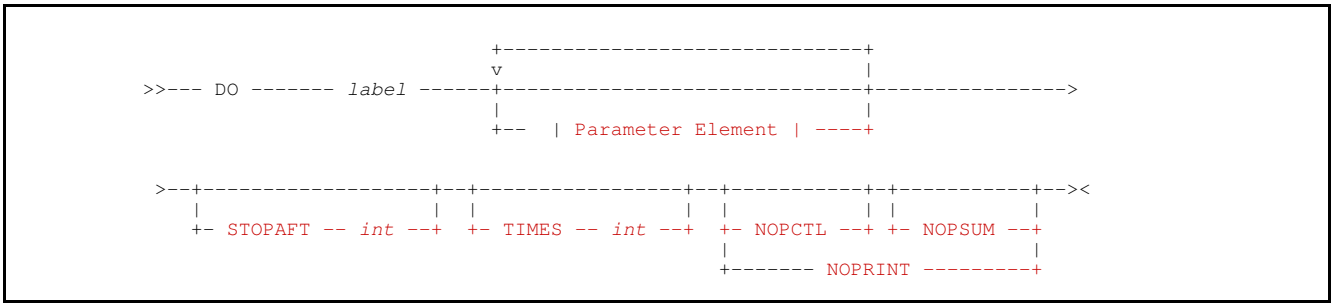
Return Codes:

0	Successful completion.
8	One of the following conditions has occurred: 1. A divide by zero has occurred. 2. The precision of *target* is not sufficient to contain the resultant value. Truncation has occurred with the loss of significant digits. 3. The first byte of a source or target field definition is within the work area but the last byte is located beyond the end of the work area buffer. The length of the field is reduced so that the last byte of the field is the last byte of the work area buffer. 4. At least one source value is treated as being of packed decimal data type but the source data is invalid packed decimal data.

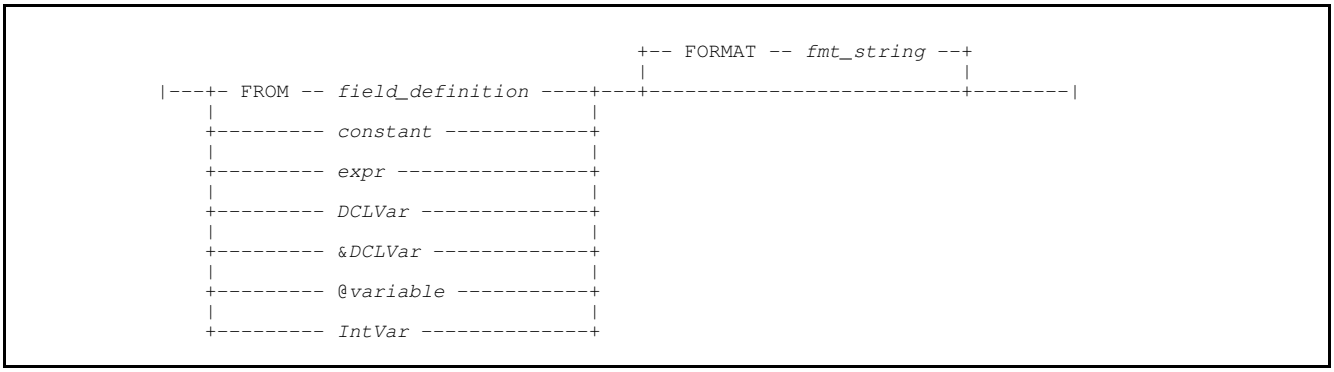
DO

Break from the standard, selection time processing sequence of control statement execution to execute a sub-routine.

Syntax:



Parameter Element:



Synonyms:

DO	GOSUB PERFORM

Description:

Execution of a DO operation will execute the sub-routine referenced by a named *label* that exists within the control statement input stream.

Control is passed to the first executable statement following *label* where selection time processing continues. The sub-routine processing ends when a RETURN operation is executed, at which time processing returns to the next executable statement following the initiating DO operation.

A sub-routine may itself execute a DO operation to execute another sub-routine, or to recursively execute the same sub-routine currently being processed.

Note that selection processing must not drop into a sub-routine. To avoid this possibility, it is recommended that sub-routines should be positioned after all the main program statements in the control statement input stream and that a GOTO operation (GET, EOJ or *label*) be included before the first sub-routine label. The GOTO operation used should either end selection processing or pass it to a control statement at the start, or within, the main processing loop.

SELCOPY sub-routines are not private procedures and so all variable and field definitions are exposed to the routine. Any changes made to variable and field definition values (including internal variable RETCODE) persist when control is passed back to the calling routine or main processing loop.

However, character values may be passed to a sub-routine as parameters. The values, identified on the DO operation as parameter element specifications, are assigned to parameter names which are local to the sub-routine. These parameter names are identifiers which follow the sub-routine *label* itself, and are assigned values in the order in which they are specified on the DO operation. Where a statement contains a sub-routine label followed by parameter names, the *label* name must be suffixed with ":" (colon).

The number of parameters passed on a DO operation need not match the number of parameters names specified following the sub-routine label. If the number of parameter values exceeds the number of parameter names, then the excess values are ignored. However, reference within the sub-routine statements to a parameter name which was not been assigned a value will set SELCOPY return code 8 and "*?*" (asterisk, question mark, asterisk) used as the missing parameter value.

If a parameter value is of numeric data type, then it will be automatically converted to its displayable character format before being assigned to a parameter name.

Parameters:

label
Specifies the statement containing the programmer defined label name, *label*, denoting the start of a sub-routine.

Parameter Element
Identifies a single parameter value to be passed to the sub-routine.

Parameter values are assigned to parameter names in the order in which they are specified.

Before being assigned to the sub-routine parameter names, parameter element values that are of numeric or character numeric data type are automatically converted to decimal character display format. Use &*DCLVar* to obtain the unformatted value of a numeric *DCLVar*.

Each parameter element may be specified in one of the following formats:

FROM *field_definition*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type.

For type 1 and 2 field definitions, the POS keyword is mandatory to distinguish the *expr* argument that follows as a field position as opposed to a parameter element arithmetic expression.

constant
A numeric constant or a quoted or hexadecimal character constant.

expr
An arithmetic expression representing a numeric value.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
The name of an @variable that has a non-null value.
If @*variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
The name of a SELCOPY internal variable.

FORMAT *fmt_string*
FMAT
FMT
FORMAT specifies *fmt_string*, a format string used as the character display template for value specified by the parameter element.

The source data in a parameter element of numeric data type is automatically converted to displayable character format (using a CVxC operation) before it is assigned to a parameter name.

For parameter elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For parameter elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the parameter element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For a parameter element specified as a numeric constant, @*variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
'SS,SSS,SSS,SS9'	Used for a constant, @*variable*, *IntVar* and *DCLVar* representing an integer value.
'SS,SSS,SSS,SS9.9999'	Used for *DCLVar* representing a rational value.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Examples:

Example 1.

The following example executes the same sub-routine, OUTRTN, to print parameter values. The first execution passes numeric values expressed as @variables (@A and @C) and an arithmetic expression (@B+@C). The second execution passes character values expressed as character declared variables.

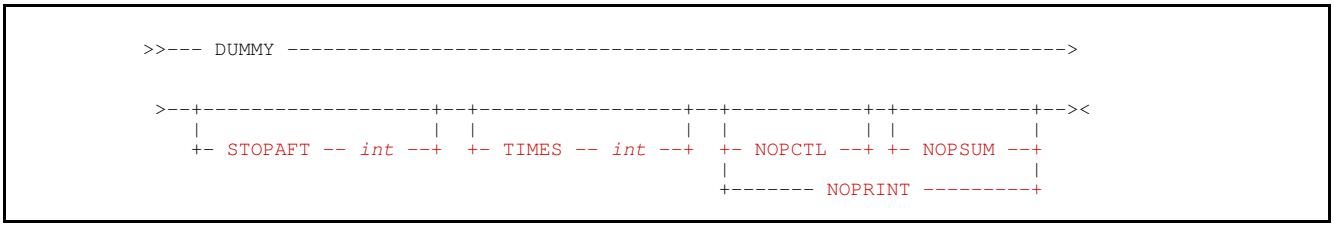
```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2016/02/03 16:01   PAGE   1
------------------------------------------------------             ----------------   --------


           DECLARE  C1     CHA  INI='Field-1'
           DECLARE  C2     CHA  INI='Field-2'

     1.    @A=1
     2.          @B=2
     3.                @C=3
                     * <---------------- TEXT ------------->    <-- P2 ->   <--- P3 --->   <-- P4 ->
     4.    DO OUTRTN  "@A FMT=99   @B+@C FMT=99   @C FMT=99 "   @A FMT=99   @B+@C FMT=99   @C FMT=99
     5.    DO OUTRTN  "C1 'and' C2   "                          C1          ' and '        C2
     6.    EOJ

         =OUTRTN:=     TEXT  P2 P3 P4 P5
          -----
     7.    SPACE 1
     8.    PRINT "--- DO OUTRTN --- "      TEXT
     9.    PRINT  P2  ' & '  P3  ' & '  P4
    10.    PRINT  P2 P3 P4 P5       * Gives RC=8 due to 5th parameter not provided by caller.
    11.  =RET=


    INPUT   SEL SEL                                                                                  1  RECORD
    RECNO   TOT ID.          1         2         3         4         5         6         7         8         9         0  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
        0     1   8 --- DO OUTRTN --- @A FMT=99   @B+@C FMT=99   @C FMT=99                                                 80
        0     1   9 01 & 05 & 03                                                                                            80
        0     1  10 010503*?*                                                                                              80

        0     2   8 --- DO OUTRTN --- C1 'and' C2                                                                           80
        0     2   9 Field-1 &  and  & Field-2                                                                               80
        0     2  10 Field-1 and Field-2*?*                                                                                 80
                    ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,....0

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1----6        1

                  =OUTRTN=
    7----9        2
   10             2      (***02 RETCD=8***)
   11             2 =ret=

***WARNING*** (SEL---10)        8 = RETURN CODE FROM SELCOPY

    ** SELCOPY/WNT 3.30.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 8. DO operation with parameters.

DUMMY

Execute a null operation.

Syntax:



Synonyms:

DUMMY	WRITE FILE=DUMMY

Description:

Executes a null operation which does nothing other than increment the selection count for the operation.

DUMMY is equivalent to WRITE DUMMY and so is treated as an output data object for which no OPEN or I/O access method service operations are actually performed.

Since end-of-job processing is automatically triggered when STOPAFT thresholds have been reached for all output (PRINT, LOG, WRITE) operations, use of DUMMY (or WRITE DUMMY) without STOPAFT may be used to force SELCOPY processing to continue until end-of-file is reached for the prime input object. e.g.

```
READ  INFILE
PRINT TYPE=M   STOPAFT=17
DUMMY                    * Prevent EOJ after 17th input record.
```

See also READ FILE=DUMMY for input of a dummy data object.

Parameters:

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

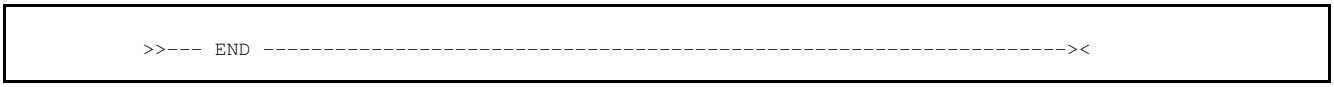
STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

END

Mark the end of control statements and the start of CARD record input.

Syntax:



Synonyms:

END	E

Description:

The END operation indicates the end of SELCOPY control statement input and is classed as one of the control statement analysis operations.

END is only required if READ FILE=CARD has been specified to input data records from the same source as the control statements. In this case, the input data records immediately follow the END operation. e.g.

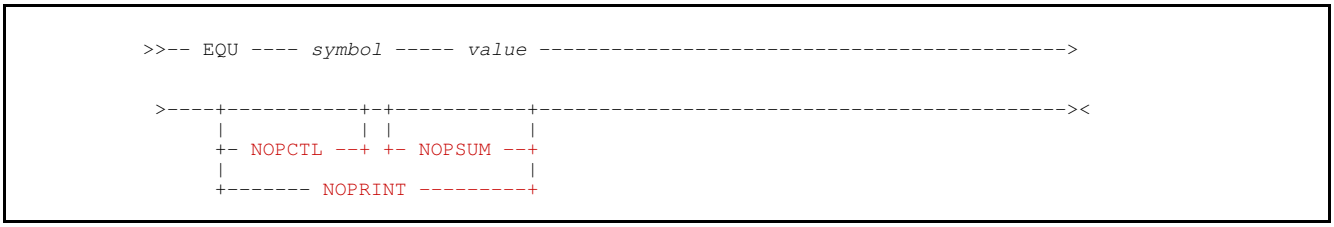
```
   READ   CARD
   PLOG
   END
Data records for CARD input go here.
/*
```

If READ CARD has not been specified, the END operation may be used instead of composite symbol "/*" (slash asterisk) in column 1, to indicate end of control statement input. (i.e. No further attempt is made to input control records). This is particularly useful when executing SELCOPY from a POSIX command shell where "/*" may be interpreted as being a generic file path.

EQU

Define an equated symbol for use in subsequent control statements.

Syntax:



Description:

The EQU operation defines an equated symbol which will be substituted with its equated value in all control statements that follow. It is classed as one of the control statement analysis operations.

Equated symbols are especially useful when referencing numeric constants and field positions in a work area, particularly if the constant or field is referenced on many control statements. Providing a meaningful name (*symbol*) for a numeric value can assist in a programmer's interpretation of the operations on which they are used. Similarly, any changes to a value that is referenced in multiple operations may easily be applied if it is equated to *symbol*, simply by updating the EQU operation.

Equated symbols are substituted in all control statements that follow the EQU operation, including statements that contain other EQU statements. Therefore, an equated *value* may itself contain an equated symbol and so nesting of equated values is possible. e.g.

```
EQU   XVAL          12     * 12
EQU   YVAL     XVAL+100    * 12+100
EQU   ZVAL     YVAL-005    * 12+100-005
```

Although not encouraged, *symbol* may contain arithmetic operator character "+" (plus) and "-" (minus). Because equated symbols are substituted during control statement analysis, if *symbol* contains one of these characters, its substitution would take precedence over its interpretation as an arithmetic expression. e.g.

```
EQU   ID             8
EQU   UNIT           15
EQU   UNIT-ID       276

PRINT FROM   POS UNIT-ID   LENGTH 20     * Position 276, not position 7 (15-8)
```

Note that no substitution will occur for *symbol* if it is referenced within a quoted character constant.

Beware when choosing a *symbol* name that it may be one of SELCOPY's keyword identifiers and, if so, may cause conflict in analysis processing of control statements that follow. e.g. POS, FROM, LENGTH are all valid *symbol* names that would likely cause a problem.

Parameters:

symbol
: The programmer defined name by which the equated symbol is identified.

value
: The equated *value* may be specified as one or more character constants which each represent a SELCOPY identifier, constant or operator statement element.

  The value specification ends at the end of the control statement on which the EQU operation is situated or before an "*" (asterisk) symbol that identifies the start of comment text.

  Note that, any unquoted literals that occur within *value*, will be upper cased. Otherwise, occurrences of *symbol* will be substituted with *value*, exactly as it is specified on the EQU operation. e.g. Apostrophes (') and quotation marks (") used to delimit quoted character constants are preserved.

NOPCTL, NOPSUM, NOPRINT
: See common parameters NOPCTL, NOPSUM, NOPRINT for details.

Example:

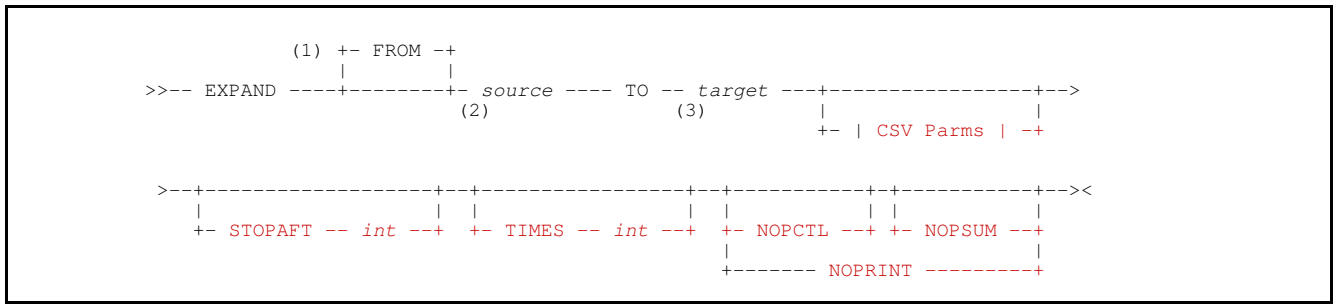
Although a better method may be to use declared variables, the following example illustrates equated symbol nesting as a method of defining a work area of length determined by the size of the consecutive field definitions used in the program control statements.

```
EQU    INREC               1    * Input Record  position.
EQU    INRECL           1000    * Input record  (maximum anticipated length.)

EQU    OUTREC    INREC+INRECL    * Output Record position.
EQU    OUTRECL           100    * Output Record (maximum anticipated length.)

EQU    TOTB    OUTREC+OUTRECL    * Total Bytes   count   (P.D. field.)
EQU    TOTBL             16    * Total Bytes   count   (P.D. field length.)

EQU    TOTR        TOTB+TOTBL    * Total Records count   (P.D. field.)
EQU    TOTRL              8    * Total Records count   (P.D. field length.)

EQU    CURL        TOTR+TOTRL    * Current Record length (P.D. field.)
EQU    CURLL           TOTRL    * Current Record length (P.D. field length.)

EQU    AVRL        CURL+CURLL    * Average Record length (P.D. field.)
EQU    AVRLL           TOTRL    * Average Record length (P.D. field length.)

OPTION  WORKLEN    AVRL+AVRLL-1  * Work Area length = sum of all field lengths.
```

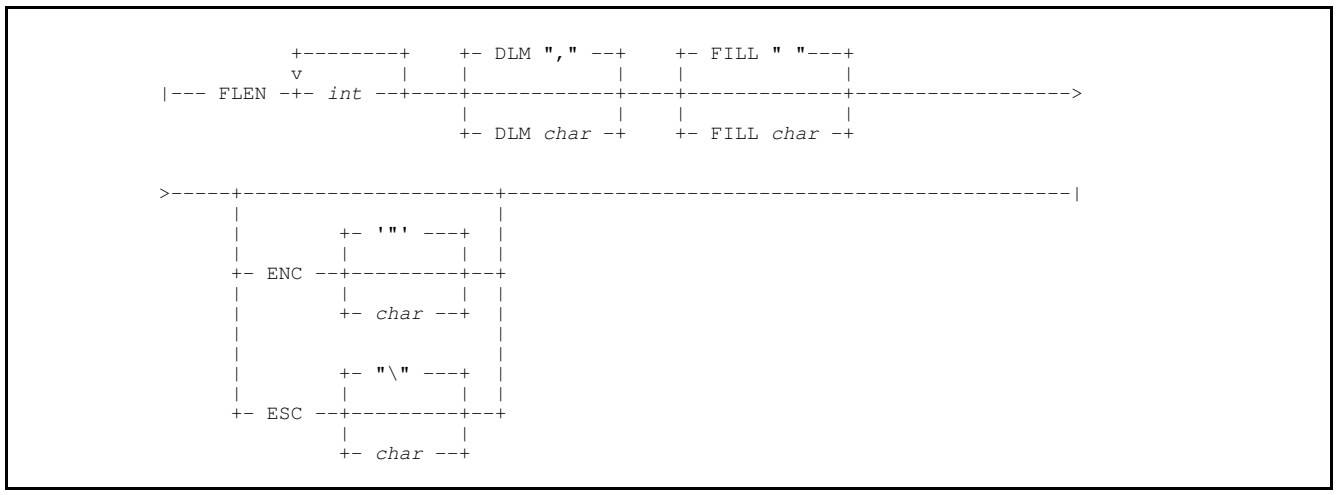
EXPAND

Expand a line of data which has previously been compressed using SELCOPY's proprietary compression algorithms. Alternatively, expand text which is in a comma separated variable (CSV) format so that values occupy consecutive fixed length fields.

Syntax:



CSV Parms:



Syntax Notes:

(1)	**FROM**	is mandatory if *source* is a field definition of type 1 (*field_pLENn*) but with keyword POS omitted.
(2)	***source***	references an area of storage specified by a character constant, a declared variable of character data type, or as a field definition of type 1 (*field_pLENn*), type 2 (*field_p1p2*) or type 3 (*field_nATp*).
(3)	***target***	references an area of storage specified by a declared variable of character data type, or a field definition of type 1 (*field_pLENn*), type 2 (*field_p1p2*) or type 3 (*field_nATp*).

Description:

EXPAND and COMPRESS operate on single lines of data (usually file object input records) to perform one of the following tasks:

1. Compress or expand data using SELCOPY's proprietary compression algorithms.
2. Compress or expand values in comma separated format (CSV).

For both types of data expansion, following execution of the EXPAND operation the value assigned to internal variable LRECL is updated to be the length of the expanded data. Furthermore, the storage containing the *source* data must not overlap the *target* storage area.

Specification of a work area is mandatory for EXPAND.

Compressed Data Processing

Where parameter FLEN is not specified, EXPAND will attempt to restore data that has been compressed using SELCOPY's proprietary data compression algorithms. See COMPRESS.

Depending on the structure of the data, it is possible that the expanded data is shorter than the compressed source.

CSV Data Processing

Data within database tables and spreadsheets can usually be unloaded into a comma separated variable (CSV) format, where each column value is separated from the following column value by a delimiter character, usually a "," (comma). In most processing environments, unloaded database data is the most common source of CSV data.

To assist in processing CSV format data, EXPAND may be used to expand each of the comma separated values into consecutive fixed length areas of a field definition or character declared variable. Once expanded, values may more easily be searched, updated and/or reformatted before, once again, being compressed into the original, or an alternate, CSV format using COMPRESS.

Parameter FLEN must be specified in order to invoke CSV data expansion instead of SELCOPY's general data decompression utility. The FLEN parameter values identify the fixed lengths of each storage area into which the data values will be expanded. When expansion occurs, each data value is copied to the next fixed length data area within the *target* area. If necessary, values will be padded to the length of the data area using the pad character, as specified by the FILL parameter.

By default, the EXPAND operation only searches for the delimiter characters, defined by parameter DLM. Leading and trailing blanks are preserved for each CSV value and special characters, such as escape characters, apostrophes or quotation marks, are treated as data and copied to the target area as part of the value. e.g. `"ABC,DEF"` will be interpreted as two fields: `"ABC` and `DEF"`

Parameter ESC should be used if the escape character exists and is not to be treated as part of the string value. Alternatively, ENC should be used if leading and trailing enclosing characters are to be stripped from CSV values when copied to the target field.

Parameters:

FROM *source*
: Specifies a character value or area of storage containing the source data to be expanded. *source* may be specified as a character constant, a declared variable of character data type, or as a field definition of type 1 (*field_pLENn*), type 2 (*field_p1p2*) or type 3 (*field_nATp*).

The complete length of character data represented by *source* will be expanded. If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL.

Keyword FROM need not be specified if *source* is specified as a character constant or a declared variable.

TO *target*
: Specifies the variable or area of storage into which the CSV values are to be expanded. *target* may be specified as a declared variable of character data type, or a field definition of type 1 (*field_pLENn*), type 2 (*field_p1p2*) or type 3 (*field_nATp*).

If FLEN is specified to indicate CSV data expansion and the length of *target* is greater than the sum of the specified FLEN values, then the last FLEN value will be used to define fields in the surplus *target* data. (See FLEN below.)

If specified as a type 1 field definition but without a length, the field will have a default length equal to that of *source*.

FLEN *int* ...
: Identifies the EXPAND operation as being for CSV data expansion. FLEN specifies an integer constant value, *int*, or a sequence of integer constants that correspond, one-to-one with the lengths of each consecutive, fixed length field in *target* area.

If the sum of the integer values specified by FLEN is lower than the length of the *target* area specification, then the last FLEN value in the sequence is used to identify the lengths of all subsequent target fields. e.g. `FLEN 30 20` indicates that the target area that will contain the first expanded value is of length 30 and target areas to contain all expanded values thereafter are length 20.

If the more FLEN values are specified than there are CSV source values, then the extraneous values are ignored.

DLM *char*
DELIM
: Specifies the delimiter character *char*, a character constant of length 1, used to separate each value in the CSV source. This character is used by EXPAND to identify each source value.

Default is "," (comma).

FILL *char*
PAD
: Specifies the fill character *char*, a character constant of length 1, used to pad values copied to the fixed length target fields.

Default is "b" (blank).

ENC *char*
: Specifies the enclosure character *char*, a character constant of length 1, to be stripped if it exists as the first and last non-blank character of a value.

Applicable only to values enclosed by this character, the leading and trailing enclosure character and any leading and trailing blanks will be stripped. Also, if 2 consecutive enclosure characters occur in the source value, then the first

occurrence will be removed. e.g. If the enclosure character is "" (quotation mark), a value `"XYZ""ABC"""` is expanded as `XYZ"ABC"` in the target field.

Parameters ENC and ESC are mutually exclusive. If ENC is specified without *char*, "" (quotation mark) is the default.

ESC *char*
Specifies the escape character *char*, a character constant of length 1, to be extracted from values in the CSV source data when it is copied to the target field area.

If 2 consecutive escape characters occur in the source value, then only the first occurrence will be removed. e.g. If the escape character is "\" (backslash), a value `\"AB\\\\C\&DEF\"` is expanded as `"AB\\C&DEF"` in the target field.

Parameters ENC and ESC are mutually exclusive. If ESC is specified without *char*, '\' (backslash) is the default.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of EXPAND for the same source data is unnecessary and so TIMES should never be used on EXPAND. See common parameter TIMES for details.

CSV Expansion Errors:

If an error is encountered during the EXPAND operation, then SELCOPY return code 8 is set, processing continues and an error message is returned in the SELCOPY list output.

Each expansion related error message includes the following element:

Field#*nn*,Len=*nn*	The sequence number and length of the target field in error.
DataLen=*nn*	The length of the value.
TotDestLen=*nn*	The length of the *target* area.

Possible errors relating to expansion of data to CSV format are as follow:

EXPND Error 01: Field#*nn*,Len=*nn* DataLen=*nn* TotDestLen=*nnn* Field data too long.
One or more of the values in the source CSV data exceeds the fixed length (FLEN value) of the target field into which it is to be expanded. Each target field for which this error is detected is filled with "*" (asterisk) characters. The target field referenced in the message text is the first one for which this error was detected.

EXPND Error 06: Field#*nn*,Len=*nn* DataLen=*nn* TotDestLen=*nnn* Destn area overflow.
An attempt was made to expand a value into a target field but that field would fall outside the target area length (TotDestLen). The target area will contain as many expanded values as will fit. The target field referenced in the message text is the first one for which this error was detected.

EXPND Error 07: Field#*nn*,Len=*nn* DataLen=*nn* TotDestLen=*nnn* Fields > Destn area.
The sum of all target field lengths provided by the FLEN parameter exceed the length of the target area. No expansion takes place and the value for LRECL is set to 0 (zero).

EXPND Error 09: Field#*nn*,Len=*nn* DataLen=*nn* TotDestLen=*nnn* Source/Destn overlap.
The source and target areas overlap. No expansion takes place and the value for LRECL is set to 0 (zero).

Examples:

The following example expands values in CSV format input records into fixed length fields.

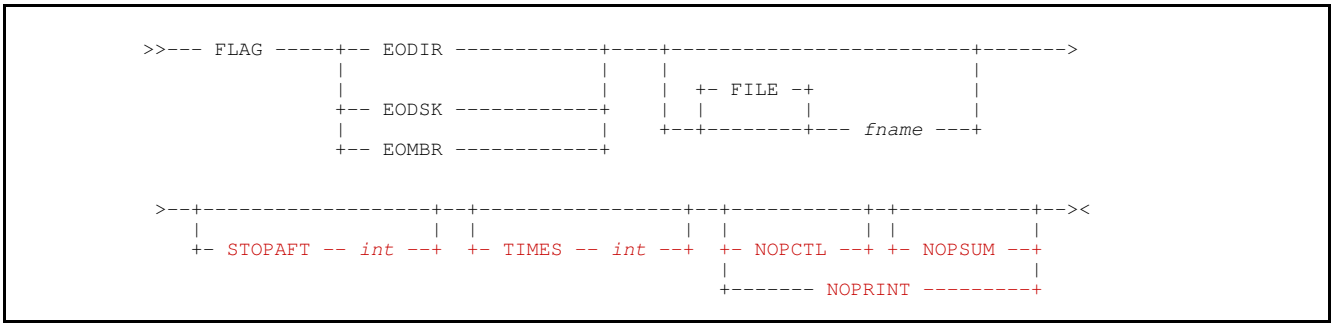
```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)                                                    2015/11/12 10:09   PAGE   1
--------------------------------------------------------                                                 ----------------   --------


          dcl scale130    cha(130)
          dcl csvvar     vcha(100)
          dcl expvar      cha(130)

          option   worklen=9999   pagedepth=999   datawidth=130

     1.   scale130        =   pos scale

     2.   print type=s  '                ' from scale130    stopaft=1

     3.   read card                        into csvvar
     4.   print type=s  'Source CSV Data: ' from csvvar

     5.   expand         csvvar            to   expvar   flen=10,5,10,10,30,20,15,10   enc
     6.   print type=s  'Expanded   Data: ' from expvar
     7.   space 1

         end


                  ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,....0....,....1....,....2....,....3
Source CSV Data: 50310201,Mrs,Gaye,Smith,"""Westminster House""",Wexham,CANTERBURY,Kent,CT3 1QP,UK
Expanded   Data: 50310201  Mrs  Gaye      Smith     "Westminster House"          Wexham              CANTERBURY     Kent      CT3 1QP   UK

Source CSV Data: 50310202,Mr,Edward,McArthur,13 Oaklands Drive,,AYLESFORD,Kent,ME12 4LB,UK
Expanded   Data: 50310202  Mr   Edward    McArthur  13 Oaklands Drive                                AYLESFORD      Kent      ME12 4LB  UK

Source CSV Data: 50310203,Ms,Lisa,Wilmot,11 St. Mary's Road,,CANTERBURY,Kent,CT1 6HT,UK
Expanded   Data: 50310203  Ms   Lisa      Wilmot    11 St. Mary's Road                               CANTERBURY     Kent      CT1 6HT   UK

Source CSV Data: 50310204,Mrs,Glynis,O'Connor,"Seafront Cottage, Whittle",Fishmonk,SEVENOAKS,Kent,TN12 4DS,UK
Expanded   Data: 50310204  Mrs  Glynis    O'Connor  Seafront Cottage, Whittle     Fishmonk            SEVENOAKS      Kent      TN12 4DS  UK

Source CSV Data: 50310205,Mrs,Patricia,Jones,Harley House,St. Joseph's Way,FOLKESTONE,Kent,CT5 3BK,UK
Expanded   Data: 50310205  Mrs  Patricia  Jones     Harley House                 St. Joseph's Way    FOLKESTONE     Kent      CT5 3BK   UK

Source CSV Data: 50310206,Mrs,Jennifer,Jales,1 Marble Cottages,Herne Road,MARGATE,Kent,CT14 4GM,UK
Expanded   Data: 50310206  Mrs  Jennifer  Jales     1 Marble Cottages            Herne Road          MARGATE        Kent      CT14 4GM  UK

Source CSV Data: 50310207,Mrs,Elizabeth,Evans,4 Matchlock Lane,,CANTERBURY,Kent,,UK
Expanded   Data: 50310207  Mrs  Elizabeth Evans     4 Matchlock Lane                                 CANTERBURY     Kent                UK

Source CSV Data: 50310208,Mr,Chris,Ainsley,18 Crofters Court,,SEVENOAKS,Kent,TN22 1PT,UK
Expanded   Data: 50310208  Mr   Chris     Ainsley   18 Crofters Court                                SEVENOAKS      Kent      TN22 1PT  UK

Source CSV Data: 50310209,Mrs,Adriana,Kay-Jones,12 Ashway,Mayhill,TUNBRIDGE WELLS,,TN8 9JK,UK
Expanded   Data: 50310209  Mrs  Adriana   Kay-Jones 12 Ashway                     Mayhill             TUNBRIDGE WELLS           TN8 9JK   UK

Source CSV Data: 50310211,Mrs,Susan,Bond,Bobble House,123 West Lane,,Kent,CT11 8XC,UK
Expanded   Data: 50310211  Mrs  Susan     Bond      Bobble House                 123 West Lane                          Kent      CT11 8XC  UK

Source CSV Data: 50310212,Mrs,Jessie,Cross,66/1 The Downs,,Folkestone,Kent,CT10 9ND,UK
Expanded   Data: 50310212  Mrs  Jessie    Cross     66/1 The Downs                                   Folkestone     Kent      CT10 9ND  UK

Source CSV Data: 50310213,Miss,Brenda,Gribble,246 Topitt Road,,WHITSTABLE,Kent,CT3 4TX,UK
Expanded   Data: 50310213  Miss Brenda    Gribble   246 Topitt Road                                  WHITSTABLE     Kent      CT3 4TX   UK


SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL          FSIZE   CI    DSN
 ------      ------      ----     -------  -----          -----   --    ---
    1           13
    2            1
    3           12 READ CARD       2048    92 U             12       C:\nbj\ca\sstest49.ctl
    4----7      12


    ** SELCOPY/WNT 3.30.002  2015/10/05  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                               ** Expiry: 14 Jul 2016 **
```

Figure 9. EXPAND CSV Format Data.

FLAG

Flag an end condition for READ DIR or READ DIRDATA input.

Syntax:



Description:

Applicable only to DIR or DIRDATA input which reads multiple directory entries or file objects with fileids that match a specified fileid mask.

FLAG sets an end of object condition for the input file, Unix-like directory or disk volume currently being processed by SELCOPY. If the prime input is not for DIR/DIRDATA or a number of READ DIR/DIRDATA operations exist for different fileid masks, then *fname* must be specified on FLAG to identify the input object stream to which the operation applies.

Parameters:

EODIR
EOD
: Applicable only to z/VM CMS SFS, Windows and Unix-like fileid masks, FLAG EODIR forces end of current input directory. All remaining entries, other than sub-directory entries, which belong to the current input directory will be bypassed. The contents of any sub-directory entries will be processed.

EODSK
EODISK
: For z/VM CMS minidisk, z/VM CMS SFS and Windows fileid masks, FLAG EODISK forces end of current input disk volume. No further directory or file entries will be read from the current disk and, if specified, processing will begin for the disk volume identified by the next drive letter.

  For Unix-like fileids, FLAG EODISK forces end of current input directory. Unlike FLAG EODIR, all remaining directory entries, including sub-directory entries, will be bypassed.

EOMBR
EOM
EOMEMb
: For any DIRDATA input, FLAG EOMBR forces early end of input file (or z/OS library member) data. The next record read will be the directory entry belonging to the next file/member to be processed.

```
READ  USER123.JCL(*)   DIRDATA              * Input z/OS JCL library.

IF DIR                                      * If directory entry.
  THEN SPACE 1                              * Skip a line in SYSPRINT.
  THEN PRINT 'Next Member: '  FROM  8 AT 1  * Print the member name.
  THEN GOTO GET                             * Get next input record. (DATA)

IF POS ANY = 'NOTIFY='                      * If JOB statement with NOTIFY.
  THEN PRINT                                * Print the record.
  THEN FLAG EOMBR                           * Flag end of data.
  THEN GOTO GET                             * Get next input record. (DIR)
```

FILE *fname*
F
: Specifies the up to 8 character file name assigned to the fileid mask used for DIR or DIRDATA input and to which the flagged condition will apply.

  If *fname* is omitted, the prime input data object is assumed by default.

NOPCTL, NOPSUM, NOPRINT
: See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
: See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of FLAG to set the same condition on the same input file object is unnecessary and so TIMES should never be used on FLAG. See common parameter TIMES for details.

Example:

The following example selects for processing all files with a file extension of "txt" that occur within the first 4 levels of directory nesting on disk volumes referenced by disk letters C, E and G respectively.

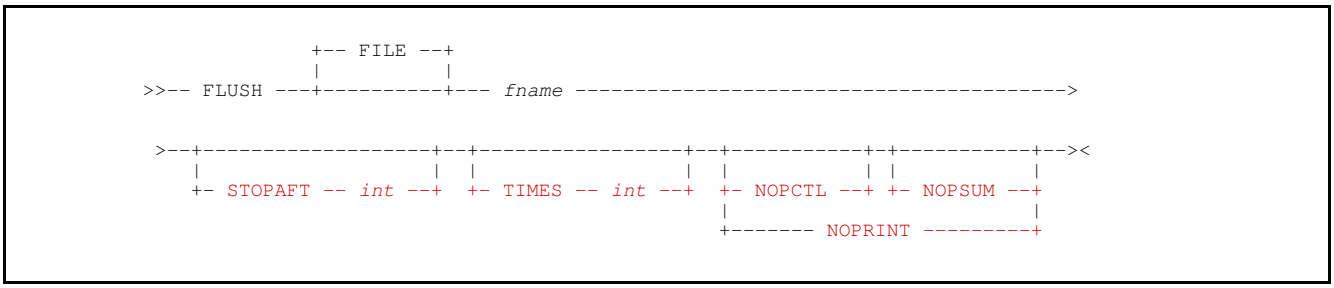
The directory entry belonging to the selected files will be printed and only the first 3 records belonging to each selected file will be read and printed. On reading a directory entry for a selected file which contains "max", then further input of files entries in the same directory are suppressed. Similarly, on reading a directory entry for a selected file which contains "tar", then further input of entries on the same disk volume are suppressed.

```
READ  'CEG:/*.txt'  DIRDATA  SUB=4   * Windows or z/VM CMS SFS fileid.

IF DATA                              * If current record is member data.
AND INCOUNT      > 2
  THEN FLAG  EOMBR                   * Force end of member for data records.

IF DIR                               * If current record is a directory entry.
THEN IF  POS ANY = 'max'
  THEN FLAG  EODIR                   * Force end of current directory.

IF DIR                               * If current record is a directory entry.
THEN IF  POS ANY = 'tar'
  THEN FLAG  EODISK                  * Force end of current disk.

PRINT                    * Print the current record, which may be a DIR entry,
                         * or 1 of the 1st 3 DATA records of each file chosen.
```

FLUSH

Flush the output buffer belonging to a file object so that the contents are physically written to the output file and buffer reset to an empty state.

Syntax:



Description:

By default, SELCOPY does not physically write the block of records stored in an output buffer to the output device until that buffer is full. i.e. On execution of a WRITE operation where the length of the output record exceeds the length remaining in the buffer.

If a buffer is not empty, execution of the FLUSH operation forces the physical write to occur before the buffer is full. Following the physical write, the file object's output buffer is reset to empty so that the next record written to the file is located at the start of the buffer.

See also parameter NEWBLK on the WRITE operation which performs the same functionality immediately prior to writing a record to the output buffer.

For RECFM VB output, a short block of records will be written to the output device with a block descriptor word (BDW) reflecting the length of the block.

FLUSH is particularly useful when output is to a Windows, Unix or Linux tape device, or a z/OS or z/VM CMS ddname that has been assigned to a tape unit.

A tape device driver will automatically insert an inter-record gap (IRG) following each block of data physically written to the tape device. Therefore, FLUSH may be used to control the size of the data block and the occurrence of an IRG.

Parameters:

FILE *fname*
F
Identifies the programmer defined file name assigned to the output file object.

The value *fname* must be specified as an unquoted literal.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Although supported as a parameter on FLUSH, TIMES is meaningless since flushing an empty buffer has no effect. See common parameter TIMES for details.

Examples:

The following working example reads an IBM emulated tape (AWSTAPE) format file and writes output to a real tape device. FLUSH is used to write inter-record gaps following each block.

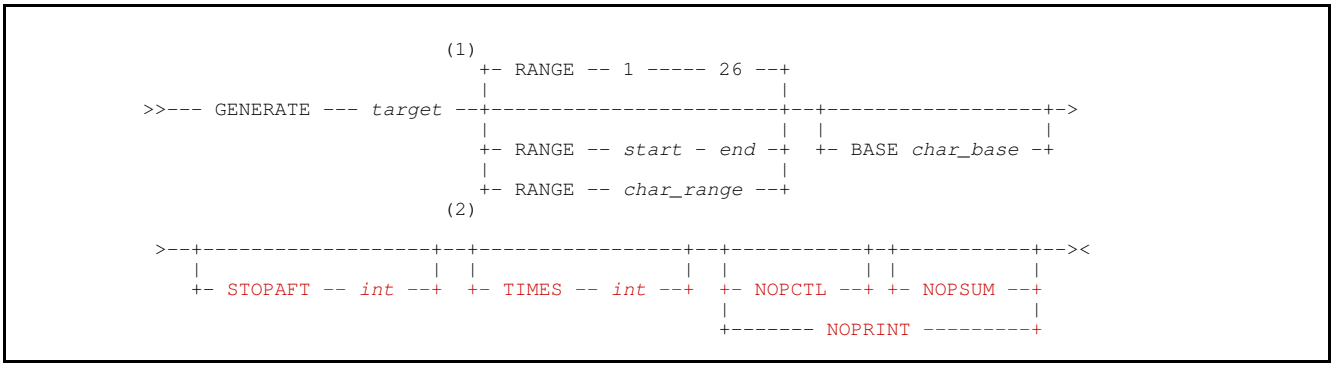
```
** c:\nbj\ca\aws2tape.ctl ***    L=002 --- 2007/04/12 17:32:36  (L05)
*
*       Title:   AWSTAPE_CTL.TXT
*        Type:   SELCOPY control statement input.
*
*   Execution:   At a system command prompt or from within a script..
*
*                     selcopy   -ctl awstape.ctl
*
* Description:   Convert AWSTAPE format file to real TAPE.
*
*       Notes:   1. UNIX tape output device *must* be non-rewinding.
*                2. Tape device driver automatically writes a Load point
*                   tape mark.
*                3. Tape device driver automatically writes a Tape Mark
*                   when output file is closed.
*                4. The tape is not rewound at end of job.
*

** Level History **
*
*
* 2007/04/12 L=002 -nbj- New Bus-Tech flags & records spanning mult AWSTAPE blocks.
* 2007/03/07 L=001 -nbj- written.


** EQUate values ***
* equ  AWSTAPE  "/home/nbj/test.aws"            * Input AWS tape fileid.
* equ  OUTTAPE  "/dev/ntibm0"                   * Output tape device.

  equ  AWSTAPE  "%1"                            * Input AWS tape fileid.
  equ  OUTTAPE  "%2"                            * Output tape device.

  equ  inb         1         * Input AWSTAPE block header (6 bytes)
  equ    obc        0        * Offset - Current  block length (Little Endian)
  equ    obp        2        * Offset - Previous block length (Little Endian)
  equ    of1        4        * Offset - Flag byte 1.
  equ      f1s    x'80'      *  Flag  - Start  of Record.
  equ      f1m    x'40'      *  Flag  - Tape Mark.
  equ      f1e    x'20'      *  Flag  - End    of Record.
  equ      f1g    x'10'      *  Flag  - Segmented Record.
  equ    of2        5        * Offset - Flag byte 2.
  equ      f2c    x'80'      *  Flag  - Bus-Tech Compression (zlib 1.1.14).
  equ      f2h    x'40'      *  Flag  - Bus-Tech Hardware Compression.
  equ      f2e    x'40'      *  Flag  - Bus-Tech Encrypted block.
  equ  inbL        6         * Input AWSTAPE block header length.

  equ  otd     inb+inbL      * Output data area.
  equ  otdL    65535         * Output data area length.  (Max Block size.)

  equ  ben     otd+otdL      * Big Endian value Binary value.
  equ  benL        2         * Big Endian value Binary value length.

  equ  mtc     ben+benL      * Tape Write Tape Mark "mt eof 1" command.
  equ  mtcL      100         * Tape Write Tape Mark "mt eof 1" command length.


  opt  worklen=mtc+mtcL      * User work area.


** Establish Tape mt Command and Output data pointer. *** (1st time only)
  pos mtc+0   = "mt -f                                  eof 1" stopaft 1
  pos mtc+6   = OUTTAPE                                        stopaft 1
  @out        = otd                                            stopaft 1

** Read Block Headers **
  lrecl       = inbL                * Length of Block Header.

  read  inaws dsn=AWSTAPE  into inb  eol=no  blksize=otdL
                                     * Read Block Header from input file.
                                     * Blksize=otdL defines an input buffer
                                     * big enough to contain maximum size block.
                                     * EOL=NO => Ignore CRLF/LF characters.

** Process Tape Mark (TM) Block Header **
  if       pos inb+of1  ones   f1m    * TM Block Header ?
   then if pos inb+obp   =   x'0000'  * No previous data block? (Previous Length=0)
         then system from mtc, mtc+mtcL-1     * Write a TM.
         then goto get                        * Get next Block Header.
         else close otape             * Close the tape output. (Automatically writes a TM.)
         then goto get                * Get next Block Header.

** Process Data Block Header **
  move  1 at inb+obc+0   to   ben+1   * Convert to 2-byte Big Endian..
  move  1 at inb+obc+1   to   ben+0   *  ..for SELCOPY.
  lrecl = benL at ben    type=b       * Length of Data in current block.

** Verify Output data area will not been exceeded by next READ. **
  if    @out+lrecl    >    otd+otdL-1
   then plog "Error.. Output Area too small. (Cannot contain next input data block.)"
   then plog "     Increase EQUated value for 'otdL', rewind the tape and re-run."
   then cancel                        * Cancel the job run.

** Process Data Block and Set output record length. **
  read  inaws              into @out eol=no   * Read data block into output pointer position.

  if    @out  = otd                  * If new output record.
   then @outL = lrecl                * Initialise output record length.
   else @outL = lrecl+@outL          * Increment the output record length.

** Write output record to tape only if end of data record. (Records may span data blocks.) **
  if       pos inb+of1  ones   f1e    * End of Record ?
   then lrecl = @outL                                  * Output record length.
   then wr    otape dsn=OUTTAPE  from otd  eol=no  blksize=otdL * Write to output buffer.
   then flush otape                  * Write contents of output buffer to file.
   then @out  = otd                  * Reset the output pointer.
   else @out  = lrecl+@out           * Increment the output pointer.

/*        ** End of SELCOPY statements. **
```

Figure 10. FLUSH Tape Output.

GENERATE

Generate a random value and assign it to a variable or field definition.

Syntax:



Syntax Notes:

(1) **RANGE** 1 26 is default only if *target* is of character data type. Otherwise, specification of a RANGE *start* and *end* value is mandatory.

(2) **RANGE** *char_range* is applicable only if *target* is of character data type.

Synonyms:

GENERATE	GEN RANDOM

Description:

The GENERATE operation will generate a random value of data type equal to that of the target variable or field definition. This value will be one of a specific range of possible values.

Parameters:

target
References the variable or field to which the generated value will be assigned. *target* may be specified as a declared variable or as a Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

The data type of *target* dictates the format of the randomly generated value. If *target* is specified as a field definition without TYPE, then a default data type of packed decimal (TYPE=P) is assumed.

BASE *char_base*
Optionally specifies *char_base* a character constant value of maximum length 4, on which the generated character or numeric value is based.

Since *char_base* is a constant value, the generated value will be the same for each execution of the individual GENERATE operation.

If a BASE *char_base* is omitted, the generated value is based upon the selection identifier assigned to the particular GENERATE operation, and the current time of day clock value. Since the time clock value is dynamic, a different value may be expected each time the individual GENERATE operation is executed.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

RANGE *start end | char_range*
Specifies a range of values from which the generated value is established.

The *start* and *end* values may each be specified as a positive or negative integer numeric constant or @variable reference. The *start* value must be less than or equal to the *end* value so that they define the limits of an ascending sequence of integers.

If *target* is of numeric data type, the RANGE parameter with *start* and *end* arguments is mandatory. The generated value assigned to *target* will be an integer that falls within this inclusive range.

If *target* is of character data type, then RANGE may be specified with *start* and *end* integer value limits or as *char_range*, a character constant.

A RANGE specification of *start*, *end* for a character data type identifies an inclusive range of values where each integer value represents a character value as follows:

1	A		8	H		15	O		22	V		29	2		36	9		43	/		50	@
2	B		9	I		16	P		23	W		30	3		37	b		44	*		51	#
3	C		10	J		17	Q		24	X		31	4		38	+		45	(		52	$
4	D		11	K		18	R		25	Y		32	5		39	-		46	)			
5	E		12	L		19	S		26	Z		33	6		40	=		47	_			
6	F		13	M		20	T		27	0		34	7		41	,		48	!			
7	G		14	N		21	U		28	1		35	8		42	.		49	:			

Specification of *start* and *end* limits that include values less than 1 or greater than 52 will generate a character value containing potentially unprintable characters.

A RANGE specification of *char_range* identifies specific array of characters from which the character value is generated. Using *char_range*, it is possible to bias selection of an individual character within the generated value by including it more than once in the *char_range* specification. e.g. `RANGE=XXA` would give a 2:3 chance that "X" occurs in any individual location within the generated character value.

The RANGE specification may be omitted if target is of character data type. If this is the case the default is **RANGE 1 26** indicating all upper case alpha characters belonging to the ISO Basic Latin alphabet (A to Z).

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Example:

The following sample output is from a SELCOPY program that uses the GENERATE operation to generate sample zoned, packed decimal, binary and character data.

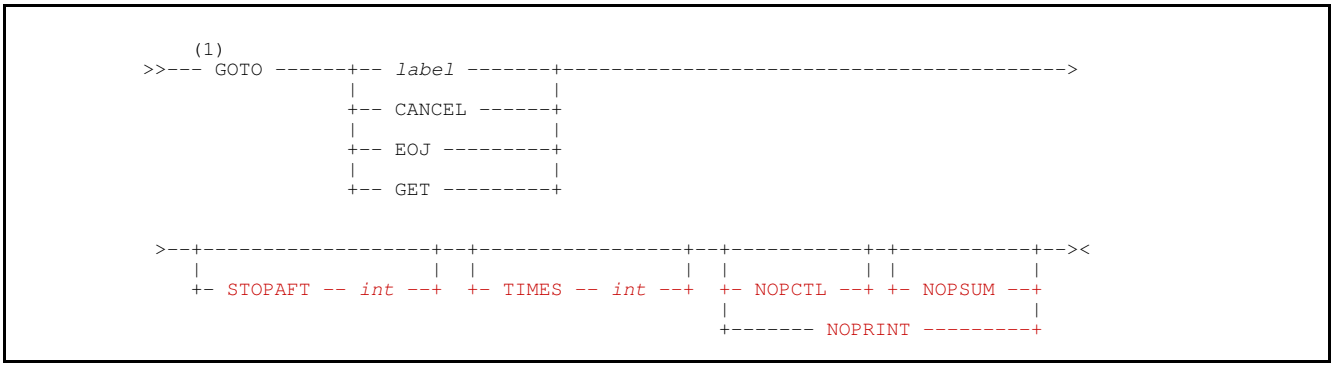
```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2016/01/08 17:42   PAGE   1
------------------------------------------------------             ----------------   --------


           OPTION DATAWIDTH=40  PRTSUM=1

     1.    PRINT 'ZONED     PACKED    CHAR      BINARY'
     2.    PRINT '-----     ------    ----      ------'

     3.    @hi  = 20         * @Variable value.

         ==LOOP==
           ----
     4.    GEN      4 AT  1  TYPE=Z    RANGE -20  @hi  BASE KGHJ
     5.    GEN      3 AT 11  TYPE=P    RANGE  51  99   BASE BB
     6.    GENERATE 9 AT 21  TYPE=C    RANGE   2   8   BASE 7777      * B to H.
     7.    GEN      4 AT 31  TYPE=B    RANGE -16  16   BASE KTWXH

         ** Now expand the data by converting to displayable hex for readability.
     8.    CVCH 4  FR 31   TO 31            * Convert Char to Hex.
     9.    CVCH 3  FR 11   TO 11

    10.    PRINT  LEN=40
    11.    GOTO LOOP   STOPAFT 5   * Loop back 5 times only.


    INPUT   SEL SEL                                          RECORD
    RECNO   TOT ID.          1         2         3         4 LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0 ------
        0     1   1 ZONED     PACKED    CHAR      BINARY         80
        0     1   2 -----     ------    ----      ------         80
        0     1  10 0000      00096C    GHHFDBEFG FFFFFFF9       80
        0     2  10 0004      00087C    BCHBDFCCD 00000006       80
        0     3  10 001N      00078C    EEEHHGHGH 00000002       80
        0     4  10 0010      00079C    FFDGFEGGE 0000000F       80
        0     5  10 001J      00092C    GBGBHCEGG FFFFFFF8       80
        0     6  10 001P      00098C    HDFGCBECB FFFFFFFB       80
                   ....,....1....,....2....,....3....,....4

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL          FSIZE   CI    DSN
 ------      ------      ----     -------  -----          -----   --    ---
    1----3        1
    4---10        6
   11             5


    ** SELCOPY/WNT 3.30.002  2015/10/05  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 11. GENERATE operation.

GOTO

Break from the standard, selection time processing sequence of control statement execution.

Syntax:



Syntax Notes:

(1) Operation keyword GOTO is optional for parameter EOJ and also for parameter CANCEL, if specified on a THEN or ELSE sub-operation of IF/AND/OR.

Synonyms:

GOTO	GO TO

Description:

By default, selection time processing executes the next statement in the sequence of control statements for which a selection identifier has been assigned or on which a condition operation (IF/AND/OR) is specified. If no executable statements follow and there exists a prime input data object which is read sequentially, then processing returns to the first executable statement. Otherwise, selection processing ends and end of job processing begins.

Where a prime input exists, the processing loop continues until an end-of-file record is read from the prime input object or a GOTO EOJ operation is executed.

This natural sequence may be overridden using a GOTO operation. GOTO supports parameters to continue selection processing at a statement containing a named *label*, to end the SELCOPY program normally or abnormally, or to return processing to the first executable statement.

Parameters:

label

Specifies the statement containing the programmer defined label name, *label*, at which selection processing will continue.

A GOTO *label* operation is usually conditional (specified on a THEN or ELSE sub-operation) based on preceding IF/AND/OR operation. It is particularly useful and necessary when a processing loop is required. e.g. The following will set an @variable (@val4) to the position of the 4th comma separated variable (CSV) value in a field.

```
  @count =   0                                    * Initialise hit count.
  @val4  = 101                                    * Field start position.

==LOOP==       * Find the 4th comma separated variable value in a field.
  IF  POS  @val4,  101+LRECL-1  =  ','   PTR @val4   * Find the next comma.
    THEN   @val4                =  @val4+1           * Character following.
    THEN   @count               =  @count+1          * Increment the hit count.
    ELSE   GOTO CANCEL                      * Unexpected so abnormal end-of-job.

  IF       @count               <  3                 * If the 1st or 2nd comma.
    THEN GOTO LABEL                                 * Repeat the loop.
 *LOOP_END*
```

CANCEL

Forces immediate, abnormal end of selection time processing.

GOTO CANCEL performs the same operation as GOTO EOJ except that SELCOPY return code 44 is set and the input buffer for the last data object read is printed in TYPE D (Dump) format as an aid in debugging.

If specified as a conditional operation on a THEN or ELSE, the GOTO operation keyword may be omitted. e.g.

```
IF  RC = 8    !THEN CANCEL
```

EOJ

Forces immediate, normal end of selection time processing and the start of normal SELCOPY end of job (EOJ) processing.

Specification of GOTO EOJ is recommended following an IF/AND/OR EOF condition on the prime input object.

If GOTO EOJ is executed before end-of-file has been reached on an input data object, the ***EOF*NOT*REACHED*** message is displayed for each READ operation on that object in the summary block of the SELCOPY list output. However, since SELCOPY has not triggered EOJ processing, return code 4 (indicating end-of-file not reached) is not set.

Similarly, execution of GOTO EOJ will suppress SELCOPY's normal check for all output selection totals being 0 (zero) and so return code 16 will not be set.

WRITE FILE=STOP is a synonym for GOTO EOJ. Similarly, the GOTO operation keyword may be omitted. e.g. The following will read and print just the first record of a file.

```
READ   INFILE   DSN='/home/user123/test_file'
PRINT
EOJ
```

GET

Skips processing of executable statements that follow the GOTO GET operation and returns processing to the first executable statement.

GOTO GET is mandatory on the last statement of the main processing loop if sub-routines (called by a DO operation) follow. This is to prevent main processing from dropping into the sub-routine control statements. e.g.

```
DO  ROUTINE     STOPAFT=1
PRINT '** Main processing **'
GOTO GET

==ROUTINE==
PRINT '** Sub-routine processing **'
RETURN
```

GG is a synonym for GOTO GET.

NOPCTL, NOPSUM, NOPRINT

See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*

See common parameter STOPAFT for details.

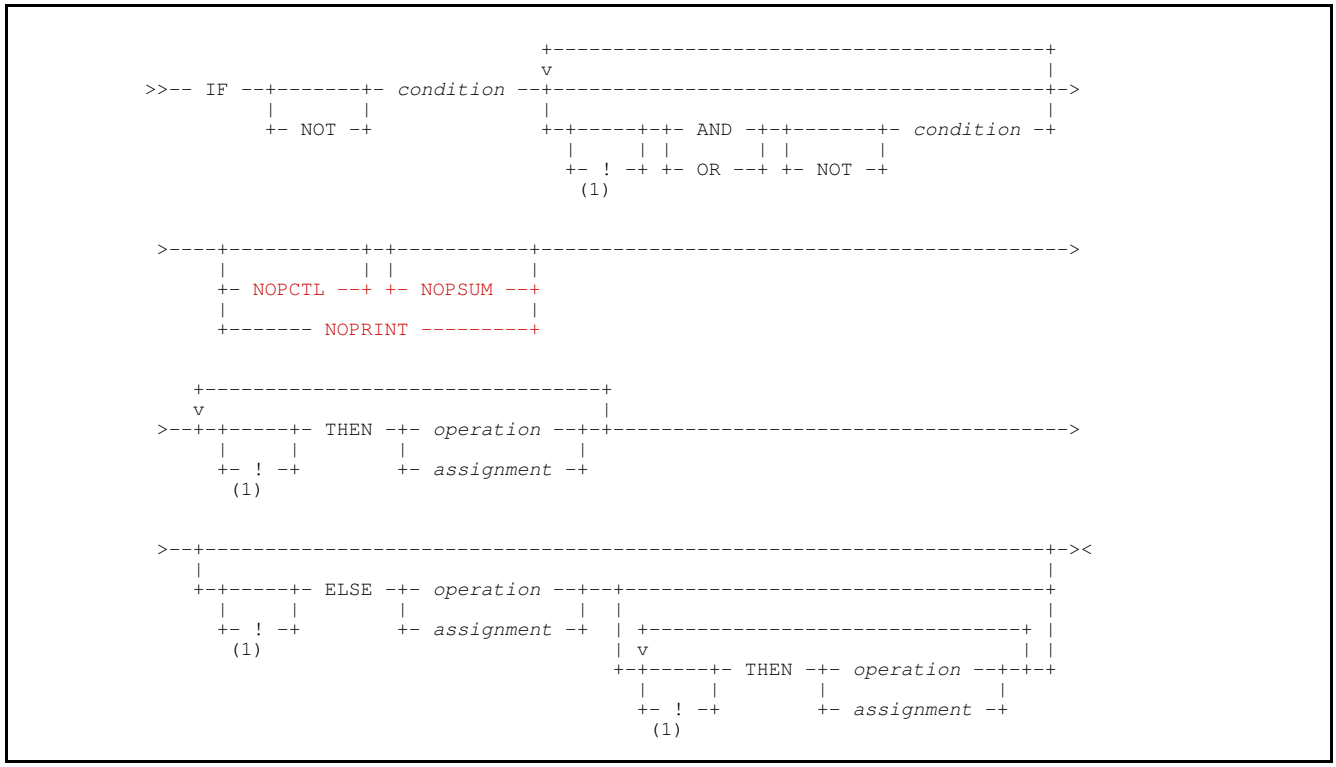
TIMES *int*

Repeated execution of GOTO is unnecessary and so TIMES should never be used on a GOTO operation. See common parameter TIMES for details.

IF/AND/OR

Test one or more conditions in order to execute one or more conditional operations.

Syntax:



Syntax Notes:

(1) **!** Keywords AND, OR, THEN and ELSE are sub-operations and so must be the first identifier specified on a SELCOPY statement. Specification of the defined statement separator character (default "!") before a sub-operation keyword is required only if it is not the first identifier. i.e. the IF operation keyword and/or sub-operation keywords exist on the same input control file record.

Synonyms:

IF	I
AND	A
OR	O

THEN	T
ELSE	EL L

THEN IF	TI THENIF
ELSE IF	LI ELSEIF

Description:

The IF operation allows conditional execution of assignment statements, selection time operations and nested IF operations.

The condition specified by logical operation, IF, together with logical sub-operations, AND and OR, define a logical expression which returns a true or false result.

A conditional sub-operation, THEN, must be specified for each operation or assignment to be executed in the event of a true result. Similarly, the operation or assignment specified on the ELSE sub-operation and each subsequent THEN sub-operation, is executed if the result is false.

The first, non-conditional operation or assignment statement that follows the last THEN or ELSE sub-operation, denotes the end of conditional operations associated with the IF operation

Processing of conditions separated by the logical AND sub-operation takes higher precedence to conditions separated by the logical OR sub-operation. Therefore, the following returns a true result if either *condition_1* and *condition_2* are both true, or *condition_3* and *condition_4* are both true.

```
IF condition_1     !AND condition_2      !OR condition_3      !AND condition_4
```

Parameters:

NOT
Reverses the result received by *condition* so that a true result becomes false and a false result becomes true.

condition
Represents one or more tests which determine the result (true or false) of the IF, AND or OR clause to which it applies. The format of the *condition* syntax depends on the type of condition being tested. Specifically, these are:

- Standard Compare
- Character Range Test
- Pattern
- Pointer
- Numeric
- List Output
- Input Object

Each condition is discussed in detail below.

AND
A sub-operation keyword specified immediately following an IF operation, an OR sub-operation or another AND sub-operation.

The AND sub-operation joins a *condition* to the preceding IF, AND or OR *condition* via a logical AND. i.e. The combined result from these conditions is true only if both conditions are true.

Note that an AND sub-operation will not be actioned if the preceding IF, AND or OR condition is false. Instead, the condition processing proceeds to the next OR sub-operation, if present. This is because the result of the group of conditions joined by logical AND operations is false and cannot be changed by further testing of subsequent AND sub-operations in the same group. The consequence is that variables that would have been set as a result of testing these conditions (e.g. DIFF, PTR=@*variable*, MATCHLEN=@*variable*/*DCLvar*) will remain unchanged.

OR
A sub-operation keyword specified immediately following an IF operation, an AND sub-operation or another OR sub-operation.

The OR sub-operation joins a *condition* to the preceding IF or OR *condition* via a logical OR. i.e. The combined result from these conditions is true if either of the conditions is true.

Note that OR sub-operations (and any AND sub-operations that follow) will not be actioned if a preceding IF or OR condition is true. In this case the overall result of the IF operation is true and cannot be changed by subsequent condition testing. The consequence is that variables that would have been set as a result of testing these conditions (e.g. DIFF, PTR=@*variable*, MATCHLEN=@*variable*/*DCLvar*) will remain unchanged.

Because processing of AND conditions take precedence over an OR condition and SELCOPY does not allow specific grouping of logically joined conditions, AND sub-operations will always be joined to preceding IF or OR operations. Therefore, if an OR sub-operation is to be joined logically to a preceding AND sub-operation, separate IF operations must be used. Where logic permits, the IF operation may be nested (conditional). e.g.
condition1 AND (*condition2* OR *condition3*) may be expressed as:

```
IF   condition_1
   THEN IF condition_2
        OR condition_3
```

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

THEN *operation|assignment*
A sub-operation keyword specified immediately following an IF operation, an AND or OR sub-operation, another THEN sub-operation or an ELSE sub-operation.

The THEN sub-operation specifies a conditional assignment, selection time operation or nested IF operation to be executed based on a true or false result obtained from all logically joined conditions specified by the IF operation.

If the THEN sub-operation occurs following an ELSE sub-operation, the *operation* or *assignment* will be executed if the overall result of the IF operation is false. Otherwise, the *operation* or *assignment* will be executed if the result is true. Specification of at least one THEN sub-operation for a true result is mandatory.

ELSE *operation|assignment*
A sub-operation keyword specified immediately following the last true result THEN sub-operation

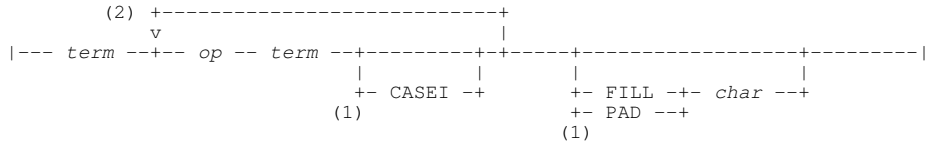
The ELSE sub-operation specifies the first conditional assignment, selection time operation or nested IF operation to be executed if the overall result of the IF operation is false. All THEN sub-operations that follow the ELSE will also be executed only if the result is false.

Specification of an ELSE sub-operation is optional and, if omitted, indicates that no action is to be taken if the overall result of the IF is false. e.g.

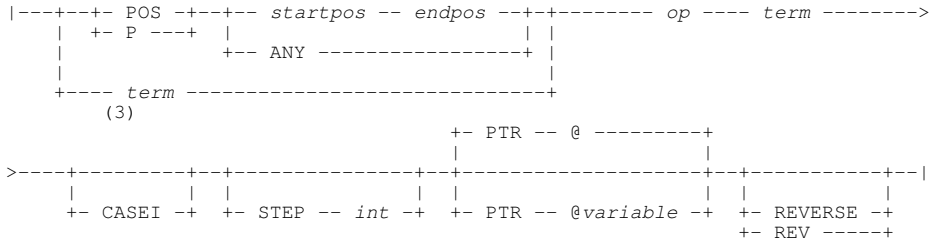
```
IF condition_1
  THEN operation_1       * Executed if condition_1 is true.
  THEN operation_2       * Executed if condition_1 is true.

  ELSE operation_3       * Executed if condition_1 is false.
  THEN operation_4       * Executed if condition_1 is false.
```

Condition Syntax:

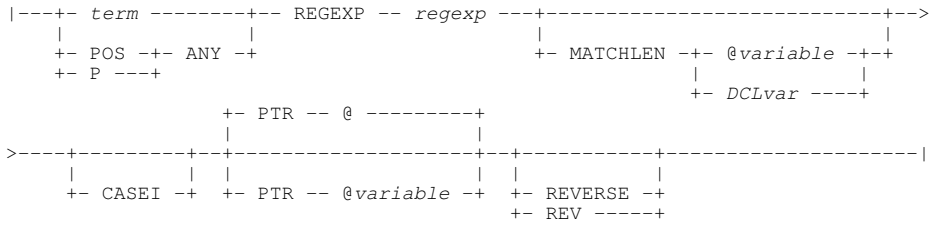
Standard Compare Condition:



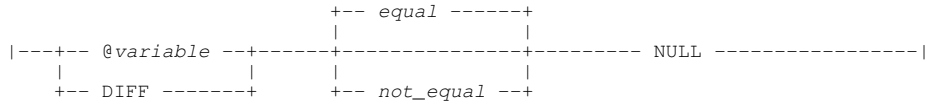
Character Range Test Condition:



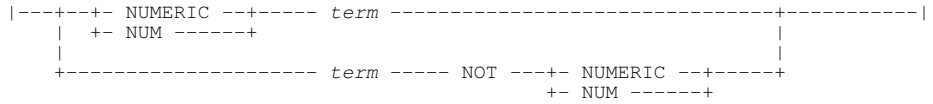
Pattern Condition:



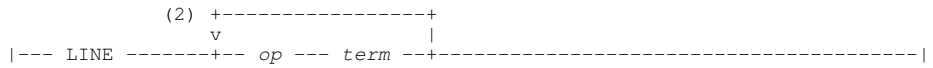
Pointer Condition:



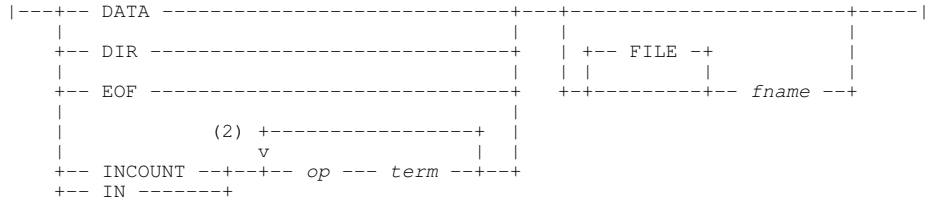
Numeric Condition:



List Output Condition:



Input Object Condition:



Condition Syntax Notes:

(1) Applicable to compare of character data only.

(2) Test conditions are performed against the first term for each *operator* and *term* combination provided none of conditions are tests for equality. (i.e. *op* is not "=", "EQ", etc.)

The result of each test condition ultimately determines the result of the IF/AND/OR *condition*. This final result is true if all test conditions are true and false otherwise (i.e. a logical AND is applied.) e.g.

```
IF   POS 21  LEN 5  TYPE=P   >= 12    <= 56    <> 27
```

This is equivalent to:

```
IF   POS 21  LEN 5  TYPE=P   >= 12
AND  POS 21  LEN 5  TYPE=P   <= 56
AND  POS 21  LEN 5  TYPE=P   <> 27
```

(3) A constant, declared variable or field definition *term* of character data type may be used to define a range of search positions only if parameter **PTR**, **STEP** and/or **REVERSE** is specified.

Condition Parameters:

term

Identifies a term of the condition expression to be compared. Unless otherwise specified, *term* may be any of the following:

◊ A constant.
◊ An arithmetic expression.
◊ A declared variable source field (&*DCLvar*).
◊ An *@variable*.
◊ An internal variable.
◊ A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.
◊ A declared variable (*DCLvar*).
◊ An IF operation keyword (DIR, DATA, EOF).

The following restrictions apply to the specification of *term*:

◊ An *@variable* or internal variable DIFF for a Pointer Condition.
◊ Internal variable LINE for a List Output Condition.
◊ Internal variable DIR, DATA, EOF or INCOUNT for an Input Object Condition.

If the first term of a condition is of character data type and parameter PTR, STEP and/or REVERSE is specified, then a Character Range Test condition is performed.

IF keyword term DIR tests the current input record for a file or library directory entry, DATA tests the current input record for a file or library member data record and EOF tests for an End-of-File condition. See description of the **Input Object** condition for details.

op

A relational or bitwise operator which identifies the compare operation performed on two terms in the condition.

The following restrictions apply to the specification of *op*:

◊ An equal or not equal relational operator for a Pointer Condition.
◊ The NOT relational operator keyword for a Numeric Condition.

Relational Operators

Relational operators specifically test for equality and/or inequalities between the 2 terms, setting a true or false condition accordingly. For terms of numeric or numeric character data type, this is an arithmetic comparison between the values represented by the terms.

For terms of character data type, a condition based on a relational operator performs an arithmetic compare on a pair of characters, one each from the same offset within each term. The value of each character is determined by the hex value assigned to it by the base encoding of the term (ASCII or EBCDIC). e.g. 'A' (EBCDIC x'C1', ASCII x'41') is less than 'B' (EBCDIC x'C2', ASCII x'42').

= b . **EQ EX EXACT**	True if terms are equal.
<> ^ ^= ¬ ¬= **NE NOT**	True if terms are not equal.
< **LT**	True if the first term is less than the second term.
> **GT**	True if the first term is greater than the second term.
<= =< ^> ¬> **LE NGT HI HIGH**	True if the first term is less than or equal to the second term.
>= => ^< ¬< **GE NLT LO LOW**	True if the first term is greater than or equal to the second term.

Bitwise Operators

Bitwise operators test the bit pattern of the first term only. All terms must be of character data type and so bitwise tests are also subject to the base encoding of the terms. The second term specifies the binary bit mask. Only those bits in the first term which correspond to 1 (one) bits in the mask are tested for being set on, bits that correspond to 0 (zero) mask bits are not tested. Results are returned for bitwise operators as follow:

ONES	True if all tested bits are set on.
ZEROS or **ZEROES**	True if all tested bits are set off.
MIXED	True if all tested bits is a mixture of bits set on and off. i.e. not all on and not all off.

If the first term length differs from the bit mask term length, then the term with the shorter length will be padded to the length of the longer term using the FILL character.

```
IF   POS 21  ONES   X'51'            * Test 1 byte  with bit mask B'0101 0001'
AND  8 AT 1  ZEROS  X'80' FILL=X'80' * Test 8 bytes with bit mask B'1000 0000'
OR   5 AT @  ONES   5 AT 101         * Test 5 bytes with work area bit mask.
```

CASEI

Applicable only to conditions that involve a character compare, CASEI indicates that character case is ignored when comparing alpha characters.

For the sake of the compare, SELCOPY takes a copy of both terms and uppercases the alpha characters in each. The compare operation is then actioned on these modified terms.

Note that, if the term is a regular expression (*regexp*), CASEI applies only to alpha characters within the expression which represent a *literal_character*.

FILL *char*
PAD

Applicable only to conditions that involve a character compare, FILL or PAD specifies the character (*char*) to be used to pad a term to the length of the longer term in the compare operation.

POS *startpos endpos*
P

By default, specification of **POS** *startpos endpos* identifies the condition as being a Character Range Test. The arguments *startpos* and *endpos* respectively identify the positions of the first and last character string to be a candidate for compare against the specified term.

For all other types of condition, this parameter would represent a *term* which is a Type 2 (*field_p1p2*) field definition.

Where **POS** *startpos endpos* is intended to represent a field definition term, parameter FILL should be specified to force a Standard Compare condition or RGX *regexp* to force a Pattern condition.

POS ANY
P

Applicable only to Character Range Test and Pattern conditions, POS ANY identifies an inclusive range of positions from 1 to the value of internal variable LRECL, in which the compare for *term* or *regexp* may be matched.

Positions outside this range are not tested so POS LRECL is the last position in which the last character of the search character string or regular expression may be found. i.e. The entire matched value must occur between POS 1 and POS LRECL.

STEP *int*

Applicable only to Character Range Test conditions, STEP specifies an integer value *int*. This is the increment value (or decrement value if REVERSE is specified) to be applied to the previous candidate string position in order to identify the position within the range of the next candidate string to be compared. See Character Range Test for more detail.

PTR *@variable*

Applicable only to Character Range Test and Pattern conditions, PTR specifies the name of the *@variable* to be assigned the position value of the matching character string. If no match is found, *@variable* will be unset (set to NULL).
If PTR is not specified, the default *@variable*, @, is used.

REVERSE
REV

Applicable only to Character Range Test and Pattern conditions, REVERSE specifies that the range of specified positions is to be searched in descending order.

REGEXP *regexp*
REGX
RGX

Specifies a regular expression in a Pattern condition.

MATCHLEN *@variable | DCLvar*

MATCHLEN is applicable to all conditions that perform a compare on character data, although is only of significant use in a Pattern condition. MATCHLEN nominates an *@variable* or declared variable (*DCLvar*) of numeric data type to be assigned a value equal to the length of the matched data.
If no successful match has occurred the value is 0 (zero).

NULL
Used to test an *@variable* or DIFF term for null (unset) status. i.e. A Pointer condition.

NUMERIC
NUM
Used to test a character *term* for being a valid numeric character data type. i.e. A Numeric condition.

FILE *fname*
Specifies the up to 8 character file name assigned to a data object in an Input Object condition.

Standard Compare Condition

Performs a character or arithmetic compare of the value defined by the first term against the value of the second and subsequent terms, and returns a true or false result based on the operator specification.

Note that the first term of a standard condition must not be a Type 2 field definition (*field_p1p2*) unless FILL is also specified. If FILL is not specified, a range test condition is implied.

All the terms must be of the same data type class, i.e. either all character or all numeric (or numeric character). Compare of date data types of different styles and source data types is not supported. However, an accurate comparison may be made if date values are converted to numeric day number values (i.e. STYLE=D) If a mixture of data type classes is used, ERROR 134 (ambiguous string or arithmetic compare) is set during control statement analysis.

If the terms are of numeric or numeric character data type, an arithmetic compare will be performed, otherwise a character compare will be performed.

Arithmetic Compare

Each term is treated as representing a signed numeric value, regardless of its source data type, length, precision and scale attributes. Therefore, the numeric terms may have different source field attributes.

Before the compare is actioned, SELCOPY converts each term as necessary so that they are of the same source data type, without loss of precision or scale.

The first value is compared, one at a time, against each of the subsequent term values and a true or false condition is returned based on the relational operator that precedes them. Note that bitwise operators are invalid for an arithmetic compare. If all comparisons are true, then the result of the condition is true. e.g.

```
IF   FLOATVAR        >=   -1.33                * Declared variable, FLOATVAR.
OR   4 AT 11 TYPE=B  >    3 AT 21 TYPE=P   <   @val-33+2
```

Character Compare

Each term is treated as representing a character string of explicit or implicit length. A byte for byte compare is performed between the underlying hex values of the characters in the first term and their corresponding characters in subsequent terms.

If the length of a character string represented by a term cannot be determined, then a length is implied which is the same as the character string term against which it will be compared. If both strings have indeterminate length, then ERROR 069 (length required) is set during control statement analysis. e.g.

```
IF   POS 101      =  'John'            * 1st term length=4 (length of "John").
```

If terms have different lengths, then, for the sake of the compare only, the term representing the shorter character string will be padded to the length of the longer string using the FILL character. Unless set by the FILL environment option, the default FILL character is blank. e.g.

```
IF   POS 7 LEN=9  =  'AS'           FILL=' '  * 2nd term padded (5 blanks).
IF   POS 1 LEN=4  =  POS 11 LEN 10  FILL='*'  * 1st term padded (6 asterisks).
```

The first term is compared, one at a time, against each of the subsequent terms using the relational or bitwise operator which precedes them. Each comparison ends when a difference is found that does not satisfy this operator, in which case the result is false, or when all characters have been exhausted (the result is true). If all comparisons are true, then the result of the condition is true.

If CASEI is specified on a subsequent term, then for the sake of the compare performed against that one term only, all alpha characters in both terms are upper cased. Alpha characters are identified for the base encoding of the character string. e.g.

```
IF   POS 1 = 'ab'  CASEI    * True if pos 1 len=2 is "ab", "aB", "Ab" or "AB".
```

If any difference is found, then, whether or not the comparison satisfies the operator, the value of internal variable **DIFF** is set, otherwise DIFF is unset (i.e. set to NULL). The value of DIFF when a difference is found, is 1 plus the displacement from the base address of the first unmatched character within the first term character string. On completion of all the term comparisons, the DIFF value will be that set or unset by the last comparison performed. e.g.

```
DECLARE  XXXVAR  CHA(10)  INI='XYZxyz'

MOD  POS 101      =  'xyz'

IF   POS 101      >  'xxx'           * True.  DIFF is set to 102.
OR   POS 101     <>  'xxx'  <> 'xyz' * False. DIFF unset by compare on 'xyz'.
OR   XXXVAR+3     =  3 AT 101        * True.  3 AT 101 padded with 4 blanks.
AND  POS 101, 103 =  'XYZ'  FILL='X' * True.  FILL prevents a range test.
AND  3 AT 101     =   xyz            * False. Unquoted literal upper cased.
AND  3 AT 101     =   xyz   CASEI    * True.  Case ignored.
AND  3 AT 101   ONES   x'70,70,70'   * True for ASCII "xyz" (x'78,79,7A')
```

Character Range Test Condition

For the character string represented by the second term (*term2*), a character compare is potentially performed against a number of character strings of the same length as *term2*, each located at successive positions in storage. i.e. A range of positions are scanned for a search string represented by *term2*.

The range of positions defines the locations of candidate strings to be compared with *term2*. However, the actual candidate strings selected for compare by the range test, depend on the following:

1. The length of *term2* and whether the range test is **inclusive** or **exclusive** of compare strings that occupy positions beyond the last position of the range.
2. The direction of the scan. i.e. Whether or not REVERSE is specified.
3. Whether or not strings are to be skipped due to the specification of STEP with a value greater than 1.

An **inclusive** range identifies all the possible start positions of the search string, *term2*. If *term2* is more than 1 character in length, the range of searched positions will include data positions that extend beyond the last position of the specified range. Therefore, the position of the last possible candidate string in the range is the last position of the range. e.g.

```
IF   POS 1, 7 = 'ABC'          * Search for 'ABC' starting in POS 1,2,3,4,5,6 and 7.
```

An **exclusive** range identifies the data positions within which the entire search string, *term2*, must be found. Therefore, the position of the last possible candidate string in the range is one where the last character of the candidate string occupies the last position of the range. i.e. the last position of the range plus 1 minus the length of *term2*.

```
LRECL = 7
IF   POS ANY  = 'ABC'          * Search for 'ABC' starting in POS 1,2,3,4 and 5 (but not 6 and 7).
```

By default, the candidate strings are compared with *term2* in ascending order of position until a match is found that satisfies the specified operator. The position of the first possible candidate string in the range is the first position of the range.

If REVERSE is specified, the order in which the candidate strings are compared with *term2* is reversed so that the last candidate string is compared first. Until a match is found, the compare operations continue for candidate strings in descending order of position.

```
LRECL = 7
IF   POS ANY  = 'ABC'     REV  * Search for 'ABC' starting in POS 5,4,3,2 and 1.
```

The next candidate string to be compared is at a position within the range which is at an increment (or decrement for REVERSE) from the previously compared candidate string position, as defined by the STEP parameter. By default, STEP=1 so no candidate strings are skipped.

```
LRECL = 7
IF   POS 1,7  = 'AB'  STEP=2       * Search for 'AB' starting in POS 1,3,5 and 7.
IF   POS ANY  = 'AB'  STEP=2       * Search for 'AB' starting in POS 1,3 and 5.
IF   POS ANY  = 'AB'  STEP=2  REV  * Search for 'AB' starting in POS 6,4 and 2.
```

Each candidate string is compared against *term2* until one is found that satisfies the relational or bitwise compare operator, or until all strings have been compared.

If a match is found, the result of the range test is true and the @*variable* specified by PTR (default @) is automatically assigned a value equal to the offset of the matching string start position from the base address. Otherwise the result is false and the @variable is unset (set to NULL).

POS *startpos endpos*
Specifies an inclusive range of ascending positions identified by *startpos endpos*.

The *startpos* and *endpos* specifications may each be expressed as an arithmetic expression (*expr*) that evaluates to a positive, non-zero, integer value. See positional syntax for a Type 2 field definition (*field_p1p2*) for more detail. e.g.

```
IF   POS 101, 101+LRECL-@len-3  =  5 AT 21          * Dynamic end position.
AND  POS 51,  100               =  'TEST'  STEP=10  * Test pos 51,61,71,81,91.
OR   POS BINVAL+3,  BINVAL+28   =  'X'              * Binary DCLvar values.
```

POS ANY
Identifies an exclusive range of ascending positions starting at position 1 and ending at position LRECL, where LRECL is the current value of the internal variable, LRECL.

Unless explicitly updated, LRECL is set to be the length of the last input data record. If the length of *term2* is greater than the LRECL value the result of the condition is false.

```
DCL  NAME5   CHA(5)   INI='James'
IF   POS ANY  =  NAME5              * Equivalent to POS 1, 1+LRECL-10
```

term1 **PTR**=*@variable* | **STEP**=*int* | **REVERSE**

If *term1* is a constant, declared variable or field definition of character data type, it may be used to define the range of search string positions only if parameter **PTR**, **STEP** and/or **REVERSE** is also specified. This is to distinguish the range test condition from a standard character compare condition.

If *term1* is a declared variable (*DCLvar*), the range test is exclusive. The start position of the range defined by *DCLvar* is the address of the variable source field (&*DCLvar*) or at an offset from that address if *DCLvar* is the first term in an arithmetic expression (e.g. *DCLvar*+@-1). The end position of the range is the last position of the character variable data.

```
DCL  IREC    CHA(100)
IF   IREC    =  'X:'    PTR=@Disk    * Equivalent to POS &IREC,    &IREC+100-2
OR   IREC+80 =  '\dir'  PTR=@Dir     * Equivalent to POS &IREC+80, &IREC+100-4
```

If *term1* is a constant or field definition, the range test is inclusive. The start and end positions of the range are defined by the limits of the constant or field definition.

Pattern Condition

For the character string represented by *term*, a single pattern matching compare is performed for the regular expression, *regexp*.

The condition tests for a match (equality) only and so no operator is involved. If an operator is specified (as for a standard condition), it is ignored without error.

Because a regular expression can represent text of varying length, *term* cannot be attributed a length from the regular expression. Therefore, the length of *term* must be implied from the specification of *term* itself.

Unless *regexp* begins with "^" (circumflex), the regular expression can match text anywhere within *term*. Therefore, a pattern condition has similar characteristics to an exclusive range test condition, setting an *@variable* when a match is found and the condition is true.

For all other character compares, the text matched by the compare operation is fixed and is determined before the condition test is performed. However, the length of text matched by a pattern condition may vary. The MATCHLEN parameter may be specified to nominate an *@variable* or *DCLvar* (a declared variable of numeric data type), which will be assigned the matched text length if a match is found.

If no match is found, the match length value assigned to *@variable* or *DCLvar* is zero (0).

```
*                         +....,....1....,....2....,....3....,.
  DECLARE DATA  CHA    INI="Source data Aa Ad Ad Ae Afff Ag Abxxx"

  IF DATA = RGX "A[~adefg]?" PTR=@HIT  MATCHLEN=@MLEN  * "Abx" at offset +32.
                                                       * @HIT->DATA+32, @MLEN=3

  IF DATA = RGX "^data"      PTR=@HIT  MATCHLEN=@MLEN  * No match.
                                                       * @HIT=0, @MLEN=0
```

Pointer Condition

The current status of the named internal variable, ***@variable*** or **DIFF** is tested for being NULL. If the status is NULL, it means the value is unset.

@variables and **DIFF** may automatically be set and unset by an IF operation. Since the value of these types of variable may explicitly be set to zero, a specific test for NULL is necessary to identify whether the variable is set before using it in another operation.

If specified, the operator must be a relational operator that tests for equal or not equal only.

```
  IF  5 AT 11         =  'Hello'               * DIFF set if false.
  AND 4 AT  1 TYPE=B  <  30                    * Arithmetic compare.

  OR  POS 21, 50      =  'World'  PTR=@World   * @World set if found.

    THEN IF  DIFF    NE  NULL   !THEN PRINT '"Hello" is missing.'
    ELSE IF  @World   =  NULL   !THEN PRINT '"World" is missing.'
```

Numeric Condition

Applicable only if *term* is of character data type, the character string represented by term is tested for being a valid numeric value. If so, *term* may be used wherever a value of numeric character data type is valid.

Valid numeric character text contains only numeric characters (0-9) and may also contain:

- One or more "," (comma) and/or blank characters use for punctuation only.
- A single "." (dot/period) character representing a decimal point.
- A single leading "+" (plus) or "-" (minus) character representing numeric sign.

List Output Condition

Perform one or more standard arithmetic compare operations against the line number of the next line to be written to the SELCOPY list output as represented by the internal variable, LINE. The value represented by *term* must be of numeric or numeric character data type and the operator used must be relational.

Input Object Condition

Tests the status of an input data object as identified by *fname*. If *fname* is omitted, the prime input data object is assumed by default.

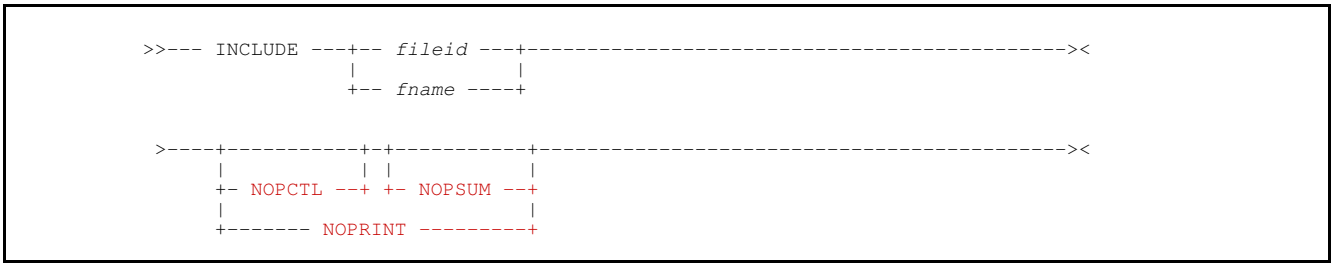
DIR and **DATA**	Applicable only to input data objects read with the DIRDATA option. The **DIR** and **DATA** input object conditions test the current input record for being a file or PDS/PDSE library directory entry, or a file or PDS/PDSE library member data record respectively.
EOF	For sequential input, tests whether End-of-File has been reached. **Warning:** Normal operation of SELCOPY selection time processing involves an implied loop of the executable control statements which automatically terminates when End-of-File is flagged following a READ operation on the prime input object. Once terminated, SELCOPY's end of job processing begins. The presence of an `IF EOF` condition for the prime input object indicates that additional processing is to occur after all prime input data has been read. To allow for this, `IF EOF` on the prime input will automatically suspend SELCOPY's termination of the processing loop until one of the following occurs: • A GOTO EOJ operation is executed. • At the start of a processing loop itreration, either the End-of-File flag or STOPAFT threshold flag has been set for all READ operations included in the program control statements. If neither of these events occur, the program will enter an infinite loop. It is recommended that, when `IF EOF` is specified for the prime input, a `GOTO EOJ` operation is included to control the point at which the loop is terminated and the start of normal end of job processing.

INCOUNT	Perform one or more standard arithmetic compare operations against the current number of input records read from the data object, as represented by the internal variable INCOUNT. The value represented by *term* must be of numeric or numeric character data type and the operator used must be relational.
	
	Note that two input record count values are maintained for data objects read using DIRDATA, one each for the directory entries and the data records. The data record count is reset to zero when a new directory record is read.
	
	The INCOUNT value for DIRDATA input applies to the type of record (DIR or DATA) that was last read from the object. To ensure a test of the correct INCOUNT value, the INCOUNT test should be performed subject to a DIR or DATA condition. e.g.
	
	`IF DIR INPDS`
	`AND INCOUNT = 10 INPDS`
	`THEN PRINT 'Processing the 10th PDS member'`

INCLUDE

Include control statements from another file.

Syntax:



Synonyms:

INCLUDE	INC

Description:

The INCLUDE operation is classed as one of the control statement analysis operations.

During SELCOPY's control card analysis, execution of INCLUDE will insert the unaltered contents of the file specified by *fileid* or *fname* into the control statement input stream immediately following the INCLUDE operation statement. SELCOPY will treat these inserted file records as part of the control statement input and include them in its control statement analysis processing.

INCLUDE allows organisation of SELCOPY control statements so that common sub-routines and program elements (e.g. EQU operations for equated symbol definitions and DECLARE operations for variable declarations) may be saved in a separate control statement file and included where appropriate.

```
selcopy  !inc ssinc01.i01  !inc ssinc01.i03  !if incount gt 20  !then eoj !end
```

Nesting of include statements is supported so that an included file can itself contain one or more INCLUDE operations.

Parameters:

fileid

Identifies the file to be included. *fileid* is a fileid clause specifying the name by which the file is known to the local system.

fileid may be specified as an unquoted literal or a quoted character constant only.

In a Windows or POSIX environment, the directories specified by environment variable, PATH, will be searched if *fileid* is a relative file path.

Notes on *fileid* specification:

1. In z/OS, where *fileid* may be interpreted as being either a z/OS data set name or an HFS/ZFS Unix-like file name, then SELCOPY will treat *fileid* first as a data set name and then, if not found, as a Unix-like file name. Similarly in a z/VM CMS environment, where fileid may be interpreted as either a CMS mini disk file specification or a BFS Unix-like file name.
2. On a z/OS system, if environment option DSNPFX is in effect (the default), then the TSO prefix (TSO/E) or ACF userid (batch) will be prefixed to *fileid* if the constant, variable or field definition value used is unquoted. Furthermore, open of a data set without a DSN prefix will be attempted if the initial attempt to open a data set with a DSN prefix fails.

   Only if this second attempt to open a data set also fails, will *fileid* be treated as a Unix-like HFS/ZFS file name as described in note 1. above.
3. A Windows or Unix-like *fileid* is not upper cased, even if specified as an unquoted literal.

fname

Applicable only to z/OS and z/VM CMS, if *fileid* is a programmer defined file name, then it will be treated as an *fname*.

The file object (z/OS data set, z/VM CMS file or HFS/BFS file path) to which *fname* is allocated will be included in the control statement input stream.

If *fname* is not allocated, then it will be treated as being part of a fileid clause specification.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

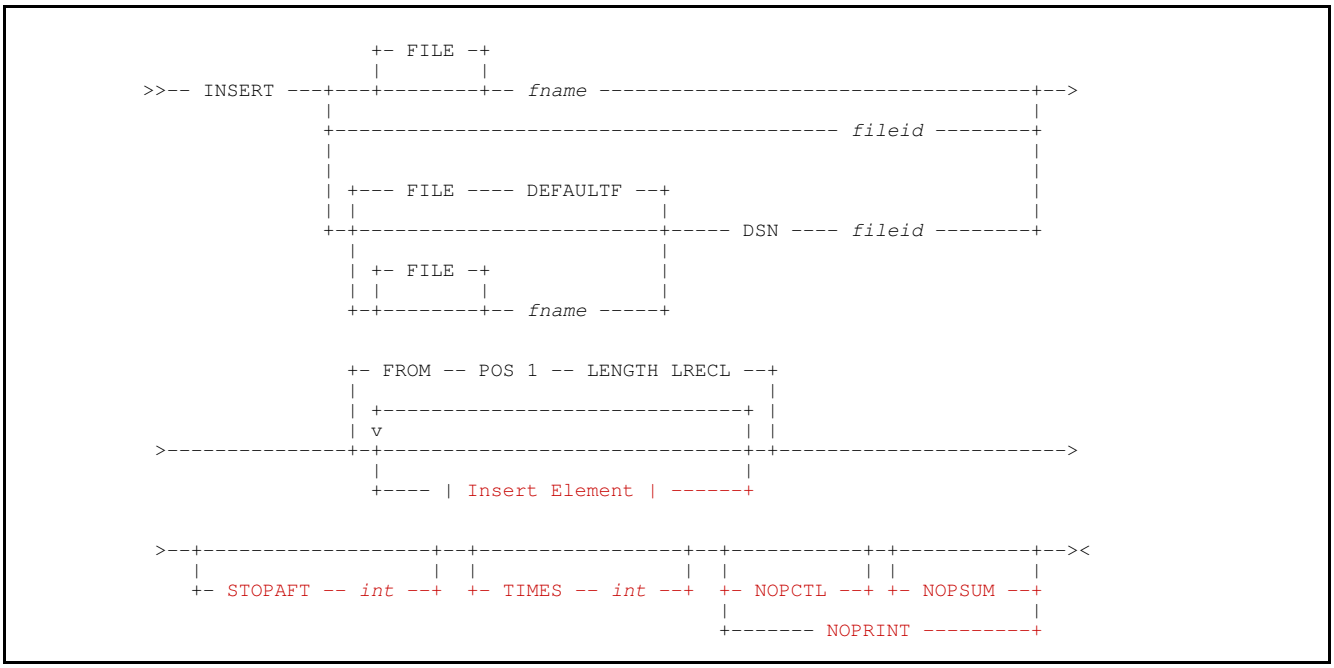
Examples:

```
INCLUDE g:\cc\slc/ctl/ssinc01.i02              * Both "/" and "\" are acceptable.
INC     ctl\ssinc01.i01                        * Relative fileid path.
INC     //serv00a/selc/ctl/equ01               * UNC fileid.

INCLUDE  DDINC                                 * z/OS or z/VM CMS fname (ddname).
INCLUDE 'CBL.SELCOPY.CTL(INCEQU01)'            * z/OS library member.
INCLUDE '/u/home/selcopy/src/ssdeclare_xfile'  * z/OS HFS full fileid path.
```

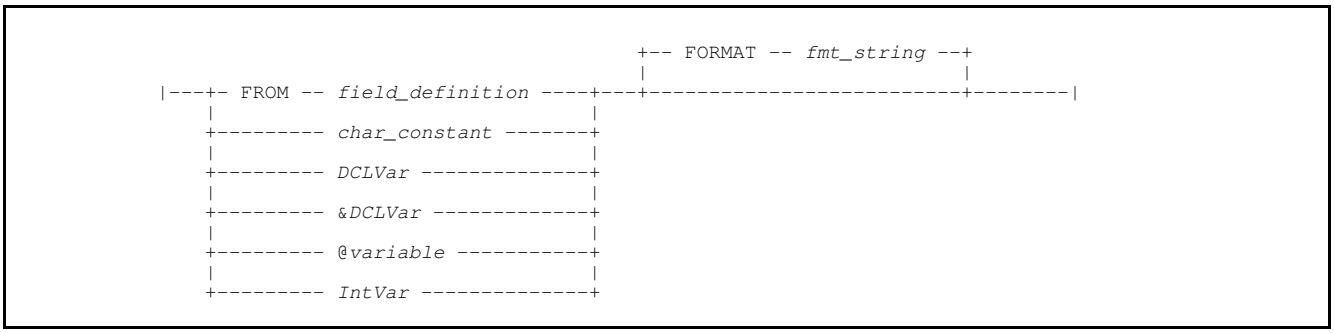
INSERT

Insert a record into a file read from an input data object source.
For ODBC table processing, INSERT is a synonym for the WRITE operation.

Syntax:



Insert Element:



Synonyms:

INSERT	INS ISRT

Description:

Insert of a record into a file object is supported for z/OS VSAM and z/VM CMS VSAM data sets with organisation KSDS or RRDS, and Micro Focus indexed and relative files. Insert of an ODBC table row is supported for an output table only and is described under the WRITE operation.

INSERT will attempt to insert a record at an eligible location within the file *fname* which may already contain data records. In order to execute the INSERT operation, the file object must first be opened for output or for input with update capability (i.e. READ with UPD).

Before INSERT may be executed for a VSAM RRDS data set or Micro Focus relative file, the READ *fname* operation must first be executed to point to the position within the file at which the record will be inserted. Furthermore, since the file must be opened for update, the UPD parameter must also be specified on a READ operation for the file. If not, ERROR 26 will be returned indicating an input/output conflict on *fname*. Note that UPD may be specified on any READ or OPEN operation for the same *fname*.

However, INSERT for VSAM KSDS data sets and Micro Focus indexed files will insert a record at the correct location within the file according to the record's key field value. Therefore, unless input processing of *fname* is required, INSERT may be executed without

having to first READ the file. If input processing is required, the UPD parameter must be specified on a READ or OPEN operation for *fname*.

Note that, if *fname* is processed for input with update the current input file position will be lost following successful execution of an INSERT operation. This is caused by the switch from direct processing for insert, back to sequential processing. Any attempt to perform a sequential read on the file immediately after an insert will return ERROR 537 (failed to reposition for sequential read). therefore, before a sequential read can be executed, a direct read is required to re-establish the input file position following an insert.

UPDATE operations, DELETE operations and INSERT operations may exist for the same data object within a SELCOPY program.

INSERT is not supported for input read via a CAT sub-operation.

Relative Record File Insert

Before a record can be inserted into a VSAM RRDS or Micro Focus relative file, a successful READ operation must be executed in order to establish an input file position before an empty slot or at the end of the file. Any attempt to insert a record at any other input file position will return an error and then terminate the SELCOPY run.

The INSERT operation will insert a record of fixed length, as defined for the input file, containing data starting at the FROM position.

Following execution of INSERT, a direct read by record (READ parameter REC or STARTREC) is required to once again establish the input file position before a sequential read can be executed.

Keyed File Insert

For VSAM KSDS data sets and Micro Focus indexed files, no read of the input file is necessary to establish an input position.

VSAM and the Micro Focus file handler will insert records into a keyed file in their correct key sequence. Therefore, records may be inserted in any key sequence. If a duplicate record error is returned following execution of an INSERT operation, then SELCOPY ends with ERROR 538 (write failed).

If records are to be inserted in ascending order of key sequence, then the WRITE operation should be used instead of INSERT. WRITE opens the file for sequential output and loads records to the file with much greater efficiency than INSERT.

The INSERT operation will insert a record containing data starting at the FROM position, with a length equal to the current value of variable LRECL. If the LRECL of the current record exceeds the defined maximum for the file, the record is truncated at the maximum length and return code 5 is set.

If the file is also being processed for input, then following execution of INSERT, a direct read by key (READ parameter KEY, KGE, STARTKEY or STARTKGE) is required to once again establish the input file position before a sequential read can be executed.

Parameters:

fileid

For file object insert where no specific *fname* has been defined on the READ operation, *fileid* may be used to identify the same file referenced on READ. *fileid* is a fileid clause specifying the name by which the input file is known to the local system.

If *fname* is specified and is already associated with a *fileid*, then re-specification of *fileid* on the INSERT operation is unnecessary.

If not specified as the DSN parameter value, then specification of FILE is invalid and the associated *fname* is derived from *fileid* as described by *fileid* for the READ operation.

fileid may be specified as an unquoted literal, a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If specified as a variable or field definition, *fileid* must be an argument to parameter DSN. Also, *fileid* specified on the READ operation for the input file, must also be a variable or field definition (i.e. a dynamic value).

DSN *fileid*

Specifies the name of the input file. See *fileid* for details.

If no *fname* is specified on an INSERT operation which uses DSN, then an *fname* of DEFAULTF is used by default.

FILE *fname*
F

Identifies the file name assigned to the input file to which records will be inserted.

fname must match the specified (or derived) file name used on the READ or OPEN operation for the file to which records will be inserted. It may only be specified as an unquoted literal.

The *fname* value may be specified with or without the FILE parameter keyword.

Insert Element
Specifies an insert element that constitutes a portion of the insert record data.

Multiple insert element specifications are concatenated with no intervening blanks and in the order in which they are specified to construct the complete insert record. The combined lengths of each of the elements define the insert record length. Note that, for keyed files, the key value will be obtained from the complete insert record at the defined key field offset.

If no insert element is specified, the insert record data defaults to a single field definition starting at position 1 with length equal to the prevailing value of the variable LRECL. (i.e. FROM POS 1 LENGTH LRECL).

All insert element values that are of numeric or character numeric data type are automatically converted to decimal character display format before being written to the insert data object (see the FORMAT parameter). Use &*DCLVar* to output the unformatted value of a numeric *DCLVar*.

Each insert element may be specified in one of the following formats:

FROM *field_definition*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type.

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL. However, if parameter FORMAT is used to convert the field data to printable hex, specification of a length is mandatory.

Although not necessary, keyword FROM may also be used before an insert element specified as a *char_constant* or a *DCLVar* of character data type.

char_constant
A quoted or hexadecimal character constant.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
The name of an @variable that has a non-null value.
If @*variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
The name of a SELCOPY internal variable.

FORMAT *fmt_string*
FMAT
FMT
FORMAT specifies *fmt_string*, a format string used as the character display template for data specified by the insert element.

The source data in an insert element of numeric data type is automatically converted to displayable character format (using a CVxC operation) before it gets written. The length of the insert element is determined by *fmt_string*, not the length of the insert element source.

For insert elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For insert elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the insert element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For an insert element specified as an *@variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
`'SS,SSS,SSS,SS9'`	Used for *@variable*, *IntVar* and *DCLVar* representing an integer value.
`'SS,SSS,SSS,SS9.9999'`	Used for *DCLVar* representing a rational value.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Example:

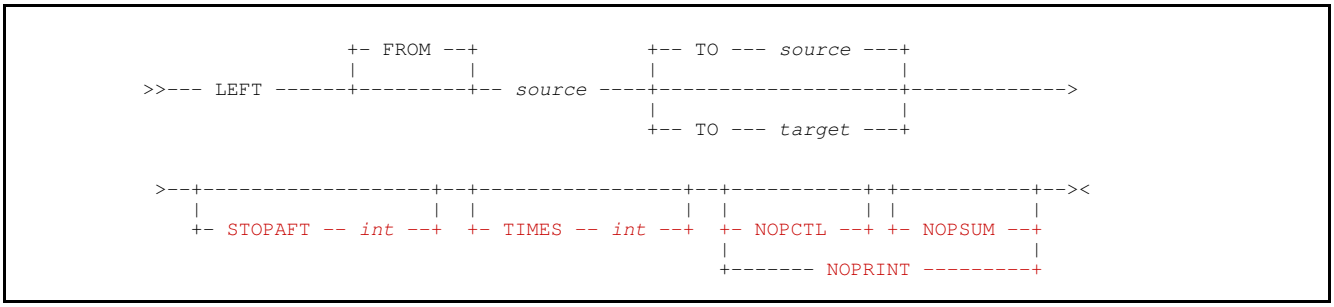
Read variable length CARD input records and insert them into a KSDS data set defined with a key field of length 7 at offset 0 of the record, i.e. KEY(0,7). Based on this key field definition, input records are in no particular order.

```
READ    CARD
PRINT
OPEN    INVS  DSN='NBJ.BASIC01.KSDS'  KSDS  UPD  STOPAFT=1  * Open for update.
INSERT  INVS                                               * Insert records.

END
0001440 Boys & Girls
0000660 21
0000040 Exile On Coldharbour Lane
0000580 Alanis Morissette: The Collection
0001550 Jagged Little Pill
```

LEFT

Left adjust text in a character source value.

Syntax:



Synonyms:

LEFT	ADJL

Description:

The text represented by the *source* character value is left adjusted and assigned to the *target* value. The *target* location may overlap storage at the *source* value location and if *target* is not specified, then the *source* value is updated.

The source text value is left adjusted in an area of storage containing all blank characters and of length equal to the *target* value field area if specified, otherwise the length of the *source* value field area.

Multiple consecutive blank characters that exist between non-blank characters in the source value text, are condensed to a single blank character in the left adjusted value. e.g.

```
POS     101 = "      xxx   yyy   zz"   *          ....,....1....,
LEFT    20 AT 101   TO  POS 1, 15      * To get: "xxx yyy zz     "
```

Parameters:

FROM *source*
References the character text value to be left adjusted. *source* may be specified as a declared variable of character data type, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition.

If TO *target* is specified, *source* may also be specified as a quoted or hex character constant.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *target* is implied. Otherwise, ERROR 69 is returned.

TO *target*
Identifies the declared variable or Type 1, 2 or 3 field definition that is the target of the operation. If *source* is a constant value, then TO *target* is mandatory.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *source* is implied.

Default is TO *source*.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

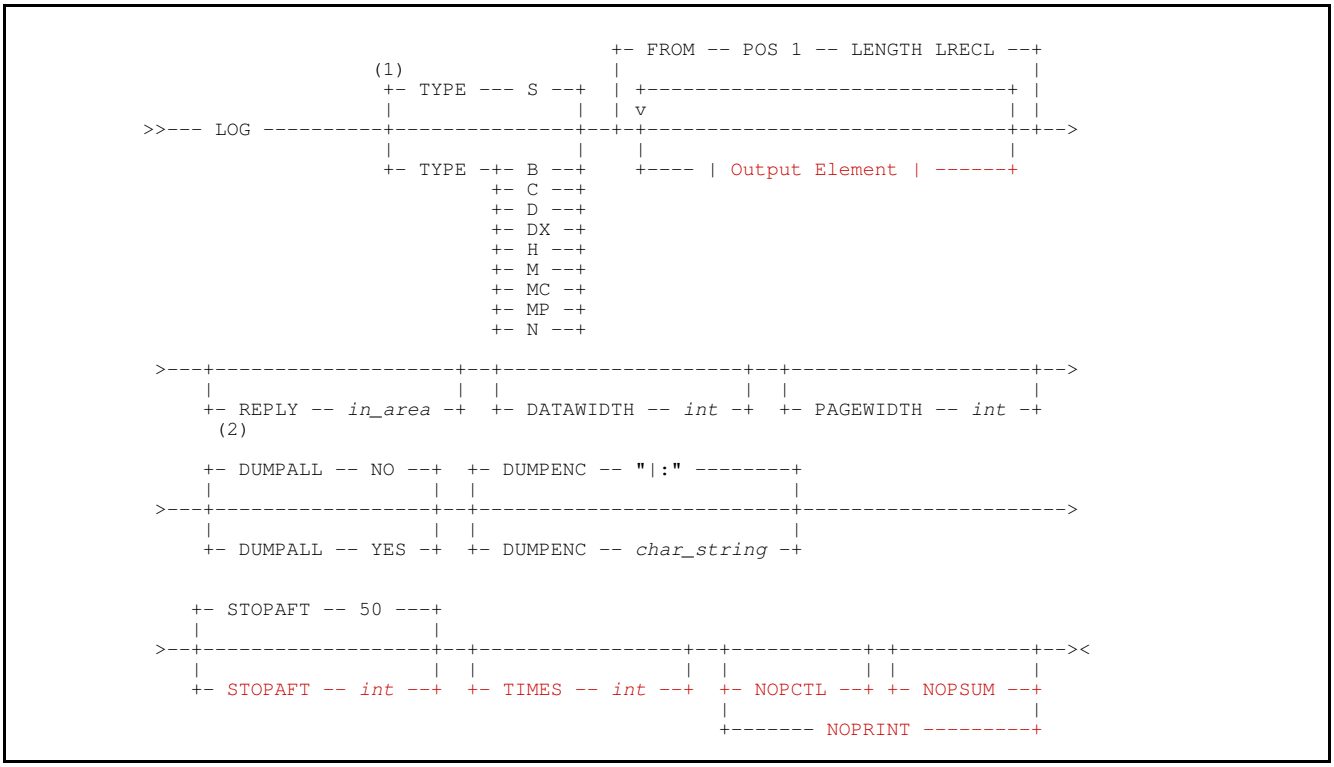
STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of LEFT for the same source and target values is unnecessary and so TIMES should never be used on the LEFT operation. See common parameter TIMES for details.

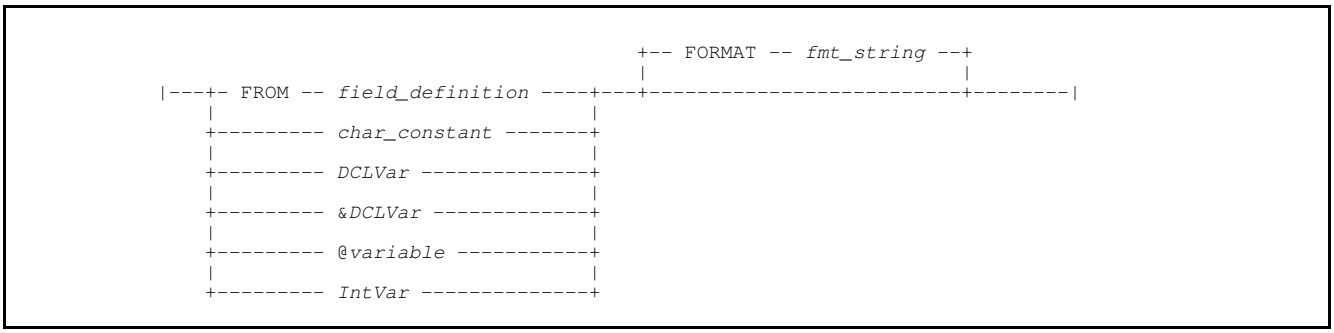
LOG

Write data to log output.

Syntax:



Output Element:



Syntax Notes:

(1) If only one output element is used, then any TYPE specification other than TYPE B may be provided following the output element specification.

(2) REPLY is supported in Windows and POSIX environments only.

Synonyms:

LOG	WTO SYSLOG

Description:

Each execution of the LOG operation will write one or more lines of data to the SELCOPY log output. The log output defaults to STDERR for Windows and POSIX environments or SYSOUT otherwise, however, output may be a file identified by input parameter `-log` *fileid*.

By default, output to STDERR, z/OS TSO/E SYSOUT and z/VM CMS SYSOUT is directed to the user's terminal.

The length of the logged data is determined by the combined display lengths of the output elements. With the exception of the header and footer lines, the format of the logged data matches the print data row that would be written to the list output print block for a PRINT operation. i.e. With the exception of TYPE S (system) output format, the logged output line will include the logged data, the prime input record number and length and also the LOG operation selection id and its execution number. See description of the SELCOPY list print block output for more detailed information.

Except when running in a z/OS batch environment, the LOG operation supports specification of parameter INTO (synonym REPLY) which allows input to the SELCOPY program following output of the logged data. If INTO is specified, SELCOPY selection processing will pause until a response is received.

Parameters:

`Output Element`
Specifies an output element that constitutes a portion of the logged data.

Multiple output element specifications are concatenated with no intervening blanks and in the order in which they are specified to construct the complete log output record. The combined lengths of each of the elements define the log output record length.

If no output element is specified, the output record data defaults to a single field definition starting at position 1 with length equal to the prevailing value of the variable LRECL. (i.e. FROM POS 1 LENGTH LRECL).

Before data is logged, output element values that are of numeric or character numeric data type are automatically converted to decimal character display format if any of the following is true:

1. The output element is a *DCLVar*.
2. Multiple output elements are specified.
3. FORMAT is included in the output element specification.

Note that &*DCLVar* may be used to output the unformatted value of a numeric *DCLVar*.

Each output element may be specified in one of the following formats:

`FROM` *`field_definition`*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of character (TYPE C) data type.

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL. However, if parameter FORMAT is used to convert the field data to printable hex, specification of a length is mandatory.

Applicable only to PRINT, LOG and PLOG operations, keyword LRECL, which usually defines the maximum output record length, is instead used as a synonym for keyword LENGTH in a type 1 field definition. Furthermore, if LENGTH is specified without a position, the default location of the field is POS 1.

Although not necessary, keyword FROM may also be used before an output element specified as a *char_constant* or a *DCLVar* of character data type.

`char_constant`
A quoted or hexadecimal character constant.

`DCLVar`
The name of a previously defined declared variable of any data type.

`&DCLVar`
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

`@variable`
The name of an @variable that has a non-null value.
If *@variable* is null, return code 8 is set and the element value of "*?*" is used.

`IntVar`
The name of a SELCOPY internal variable.

`FORMAT` *`fmt_string`*
`FMAT`
`FMT`
FORMAT specifies *fmt_string*, a format string used as the character display template for data specified by the output element.

The source data in an output element of numeric data type is automatically converted to displayable character format (using a CVxC operation) before it gets printed. The length of the output element is determined by *fmt_string*, not the length of the output element source.

For output elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For output elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the output element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For an output element specified as an *@variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
`'SS,SSS,SSS,SS9'`	Used for *@variable*, *IntVar* and *DCLVar* representing an integer value.
`'SS,SSS,SSS,SS9.9999'`	Used for *DCLVar* representing a rational value.

DATAWIDTH *int*
DW

Applicable to all types of log format other than system (TYPE S) or dump (TYPE D or DX), DATAWIDTH specifies the width of the printed data column in the print block. i.e. the length of output element data to be written to each line of the log output.

DATAWIDTH is an environment option which is established during control statement analysis and applies to all PRINT and LOG operations executed during the SELCOPY program run. See OPTION DATAWIDTH for further details.

The default is the value set by OPTION DATAWIDTH in the SELCNAM file, otherwise a value of 100 is used.

DUMPALL YES | **NO**

Applicable to dump (TYPE D or DX) format only.

DUMPALL determines whether or not second and subsequent consecutive lines of dump format output that contain identical data to the first line, are to be condensed and displayed as a single output row containing the literal "=same=" followed by a parenthesised count of the number of condensed lines.

For the LOG operation on which it is specified, DUMPALL YES will output all line containing duplicate data, DUMPALL NO will condense these lines. Note that DUMPALL is a synonym for DUMPALL YES.

The default value is set by environment option, DUMPALL. Otherwise, the default is NO.

DUMPENC *char_string*

Applicable only to dump format which includes character representation of the data (TYPE D).

For the LOG operation on which it is specified, DUMPENC specifies *char_string*, a character constant of length 1 or 2, which identifies the characters used to enclose the character representation of the dump format output.

1. The first (or only) character defines the enclosing character used on logged lines that have not been condensed to a single line. (See option DUMPALL above).

2. The second character defines the enclosing character used on logged lines that have been condensed to a single line.

The default value is set by environment option, DUMPENC. Otherwise, "|" (or symbol) and ":" (colon) are used as the first and second characters respectively.

NOPCTL, NOPSUM, NOPRINT

See common parameters NOPCTL, NOPSUM, NOPRINT for details.

PAGEWIDTH *int*
PW

Although the LOG operation does not output data to the SELCOPY list output, the PAGEWIDTH is still relevant to data logged in dump (TYPE D or DX) format. For dump format only, PAGEWIDTH restricts the length of printed data written to each line of the print block.

See description of TYPE=D in the print block output for details on how PAGEWIDTH affects dump format print.

PAGEWIDTH is an environment option which is established during control statement analysis and applies to all PRINT and LOG operations executed during the SELCOPY program run. See OPTION PAGEWIDTH for further details.

The default value is set by OPTION PAGEWIDTH in the SELCNAM file, otherwise a value of 132 is used for POSIX environments and 133 for z/OS and z/VM CMS environments..

REPLY *in_area*
INTO

Applicable only in Windows and POSIX environments, presence of the INTO parameter will pause SELCOPY processing following execution of the LOG operation, pending input from the STDIN input stream which is usually assigned to the input terminal.

REPLY specifies *in_area* which identifies a length and location in storage into which the inputted text string will be returned without translation. *in_area* may be specified as a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If *in_area* is a field definition, the following is true:

1. For a type 1 field definition, LRECL may be used as a synonym for LENGTH.
2. If a single numeric argument, then this value represents the length of the input field, not its position. The default field position is POS 1.

If the length of *in_area* is greater than the length of the inputted text string, then the input text will be left adjusted within *in_area* and padded with blanks. Therefore, a null string will result in an *in_area* containing all blanks.

The actual length of the input text string last entered following a LOG operation with REPLY, is saved as a 4-byte binary integer value in the internal field UXREPLYL.

STOPAFT *int*

See common parameter STOPAFT for details.
Default is STOPAFT 50.

TIMES *int*

See common parameter TIMES for details.

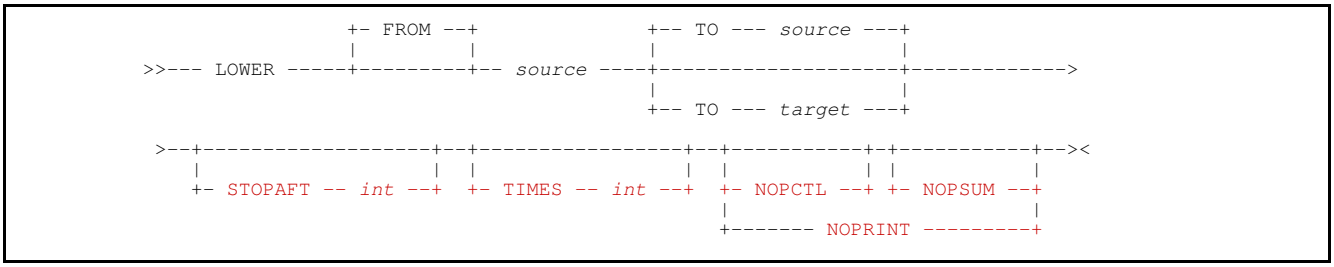
TYPE **B** | **C** | **D** | **DX** | **H** | **M** | **MC** | **MP** | **N** | **S**
TY

Identifies the type of print output to be used when writing the logged data. See PRINT for detailed information on TYPE options.

LOWER

Lower case alpha text in a character source value.

Syntax:



Description:

All upper case alpha characters belonging to the ISO Basic Latin alphabet that occur within text represented by the *source* character value, are lower cased and the value assigned to *target*. The *target* location may overlap storage at the *source* value location. If *target* is not specified, then the *source* also becomes the target of the operation and its value is updated.

The interpretation of a character as alpha depends on the local code page used by the system on which SELCOPY executes. In all ASCII and EBCDIC code pages, the 26 upper case Latin alpha characters (A to Z) are at invariant code points. i.e. ASCII [x'41'-x'5A'] and EBCDIC [x'C1'-x'C9', x'D1'-x'D9', x'E2'-x'E9']

Each occurrence of an upper case alpha character will translate to its equivalent lower case representation (a to z). These characters also exist at invariant code points. i.e. ASCII [x'61'-x'7A'] and EBCDIC [x'81'-x'89', x'91'-x'99', x'A2'-x'A9']

```
DECLARE  REF    CHAR      INI='WILLIAMS, Brian --- 32% '
LOWER    REF
PRINT    REF                           * Prints: "williams, brian --- 32%"
```

The LOWER operation is a synonym for the TRAN operation with parameter LOWER.

If lower casing is required for extended Latin characters or characters belonging to other alphabets, then the TRAN operation must be used. The translate table referenced by the TRAN operation would need to identify each character's lower case code point at its upper case code point location within the table.

Parameters:

FROM *source*
References the character text value in which alpha characters will be lower cased. *source* may be specified as a declared variable of character data type, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition.

If TO *target* is specified, *source* may also be specified as a quoted or hex character constant.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *target* is implied. Otherwise, a length equal to the prevailing value assigned to variable LRECL is used.

TO *target*
Identifies the declared variable or Type 1, 2 or 3 field definition that is the target of the operation. If *source* is a constant value, then TO *target* is mandatory.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *source* is implied.

If the length of *target* is less than that of *source*, then the value assigned to *target* is truncated on the right. If its length is greater than that of *source*, then the *source* length is used. In this case, text assigned to *target* that exists at locations beyond the length of *source*, remains unchanged following the UPPER operation. (i.e. No padding occurs.)

Default is TO *source*.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

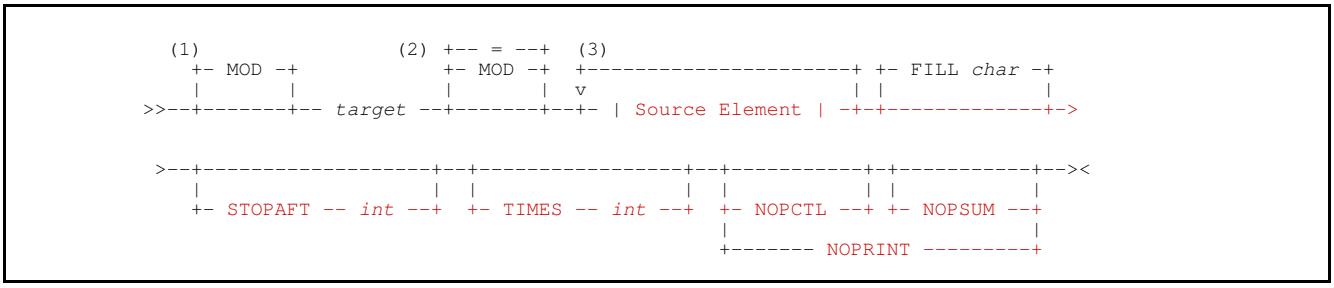
STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of LOWER for the same source and target values is unnecessary and so TIMES should never be used on the LOWER operation.
See common parameter TIMES for details.

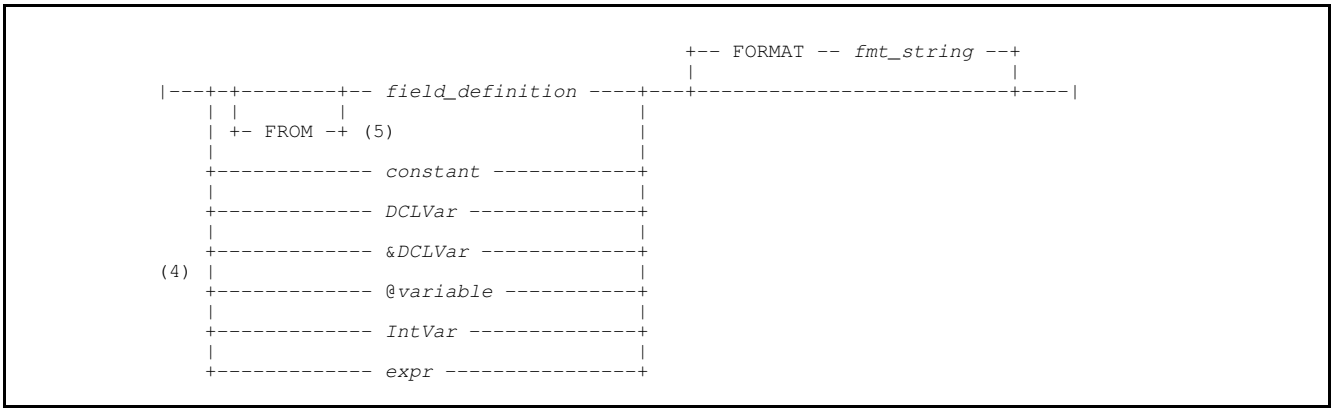
MOD

Modify the value assigned to a variable or field definition.

Syntax:



Source Element:



Syntax Notes:

(1) Operation keyword MOD must **not** be specified if *target* is an internal variable (e.g. LRECL or RETCODE) or an *@variable*. However, keyword MOD is mandatory if *target* is a Type 3 (*field_nATp*) field definition. For all other specifications of *target*, keyword MOD is optional.

(2) Parameter keyword, MOD, or the delimiter character "=" (equals) is mandatory if *target* is a field definition without a length value.

(3) Multiple source elements are supported only if *target* is of character data type.

(4) *@variable*, *IntVar* and *expr* source elements are valid only if *target* is of numeric or numeric character data type.

(5) FROM is mandatory if *field_definition* is one of a number of specified source elements. Otherwise it should not be used.

Synonyms:

MOD	ASSGN LET

Description:

The MOD operation is an assignment statement, used to assign a value directly to a variable or field definition.

The *source* and *target* specifications may be of different data types. If so, the MOD operation will automatically attempt to convert the source value to the *target* data type.

If the *target* variable or field definition is of character data type, then multiple source element values may be specified. Multiple source element values are concatenated as character strings with no intervening blanks, and together define the value assigned to *target*. e.g.

```
DECLARE  PDVAL    DEC(8,2)   INI='-8,637.44'   FMT='999,999.99 CR'

MOD POS 101  =  '---'  FROM 5 AT 22 '---  : '  PDVAL  ':  '  FROM 4 AT 1
```

If the length of a character type *target* is greater than that of the assigned value, then the value is padded on the right to the length of the target using the character specified by the FILL parameter. If the length of the target is less than the assigned value, then the value is truncated on the right.

For character value assignment, the source value is copied from left to right to the *target* location. If the field belonging to the *target* value overlaps the field belonging to a source value at any location beyond the first position of the source field, then the effect is as if characters are copied one at a time from the source to the *target* fields. i.e. A character that has been copied to the target field becomes a source field character that is copied later in the execution of the operation. e.g. In the following, characters "Ab" at positions 1 and 2 are copied to positions 3 and 4 respectively before characters at positions 3 and 4 are copied to positions 5 and 6, etc.

```
MOD POS 1, 10  =  'Abc'     FILL='X'   * Source: 'AbcXXXXXXX'
MOD POS 3      =   POS 1 LENGTH 6      * Target: 'AbAbAbAbXX'
```

This behaviour provides a useful technique by which character data may be propagated within a field.

If the *target* variable or field definition is of numeric data type, then the assigned value must have a numeric interpretation. The numeric value is then converted to the data type of the target.

Parameters:

target
References the variable or field to which the value will be assigned. *target* may be specified as a declared variable, internal variable or @variable, or as a field definition of any type.

If specified as a Type 4 (*field_pFMT*) field definition, the *target* is of numeric character data type.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then, regardless of a TYPE specification, *target* is treated as being of character data type and the total length of the source elements is implied. Otherwise, ERROR 69 is returned.

Source Element
Specifies a source element value to be assigned.

If *target* is of numeric or numeric character data type, then only one source element may be specified. The source element may be an *@variable*, an internal variable (*IntVar*), an arithmetic expression (*expr*) or a constant, declared variable or field definition of numeric or numeric character data type.

If *target* is of character data type, then multiple source elements may be specified to define a single value with a length equal to the combined lengths of all the source element values. Multiple source element values are concatenated in the order in which they are specified.

Before a character value is assigned to *target*, source element values that are of numeric or numeric character data type are automatically converted to decimal character display format if any of the following are true:

1. The output element is a *DCLVar*.
2. FORMAT is included in the source element specification.

Note that &*DCLVar* may be used to output the unformatted value of a numeric *DCLVar*.

Each source element may be specified in one of the following formats:

FROM *field_definition*
FR
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type. If *target* is of numeric or numeric character data type, then the field definition must be of numeric or numeric character data type.

If specified as a type 1 field definition but without a length, then the length of *target* is implied.

Specification of the FROM keyword is mandatory if *field_definition* is just one of a number of source elements. Otherwise, it should not be used.

constant
A constant representing a character, numeric or numeric character value.

If *target* is of numeric or numeric character data type, a hex character constant source value is treated as being a hex binary integer constant.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
Supported for numeric or numeric character *target* values only, *@variable* is the name of an @variable that has a non-null value.
If *@variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
Supported for numeric or numeric character *target* values only, *IntVar* is the name of a SELCOPY internal variable.

expr
Supported for numeric or numeric character *target* values only, *expr* is an arithmetic expression.

FORMAT *fmt_string*
FMAT
FMT
For character *target* values only, source elements of numeric or character numeric data type are automatically converted to displayable character format (using a CVxC operation) before the assignment is performed.

FORMAT specifies *fmt_string*, the numeric format string used as the character display template for data specified by the source element. The length of the source element is determined by *fmt_string*, not the length of the element's source field.

If the source element is specified as a *DCLVar*, a default *fmt_string* is used if no FORMAT parameter is specified. This is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
'SS,SSS,SSS,SS9'	Used for a *DCLVar* which represents an integer value.
'SS,SSS,SSS,SS9.9999'	Used for a *DCLVar* which represents a rational value.

FILL *char*
PAD
Applicable only to assignment of a value to a *target* of character data type. FILL defines the pad character (*char*) to be used when the length of target is less than the total length of the source value.

The pad *char* may be specified as a quoted or hex character constant of length 1.

The default is the blank character.

MOD
=
The MOD parameter keyword may be used to distinguish the *target* specification from the source element(s). Its use is mandatory if *target* is specified as a Type 1 (*field_pLENn*) field definition without a length. e.g. In the following operation, parameter MOD is necessary to prevent SELCOPY from interpreting *target* as a Type 2 (*field_p1p2*) field definition:

```
 POS 5, 10

  MOD  POS 5    MOD   10  AT  101    *  Target field is:  POS 5 LENGTH 10.
```

In all SELCOPY control statements, the "=" (equals) symbol is treated as a blank delimiter. However, for the MOD operation only, it may also be interpreted as a synonym for parameter MOD.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Examples:

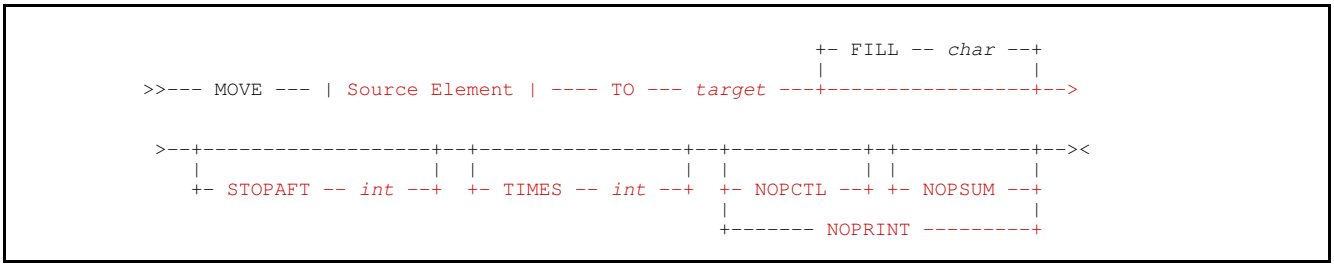
```
  DECLARE  ID         BIN(4)                    * Binary integer variable.
  DECLARE  PRICE      DEC(8,2)                  * Decimal fixed point variable.
  DECLARE  COMPF      FLT(8)  BIN               * Double precision floating point.

  DECLARE  INV_ADDR1 CHA(50)                    * Character variable.
  DECLARE  COMPANY   CHA(60)                    * Character variable.
  DECLARE  CHA_VAL   CHA      FMT='ss,ss9'      * Numeric character variable.

  OPTION   WORKLEN 2000                         * Work area buffer.

  MOD  POS   1    'ABC Holdings Ltd'            * Character field (length=16).
  MOD  4 AT 31  TYPE=B  =   255                 * Binary integer field.
  MOD  8 AT 41  TYPE=P  =   4184                * Packed Decimal integer field.

       PRICE     =  '4,385.50'                  * Fixed point decimal constant.
  MOD  COMPF     =  '-2.484682e15'              * Constant value with exponent.
```

```
      POS 51,71  'Ms Antonia Harding'  FILL='-'  * Padding    (18 into 20).

MOD   COMPANY   =  POS 1, 20                     * Padding    (20 into 60)
      INV_ADDR1 =  COMPANY                       * Truncation (60 into 50).

      @VAR      =  4 AT  31 TYPE=B               * @variable value.

MOD   ID        =  POS 41 LENGTH=8 TYPE=P        * Packed decimal to Binary.

      @VAR      =  @VAR+ID-100                   * Arithmetic expression.
      LRECL     =  55+@VAR                       * Internal variable value.
      RC        =  33                            * Set SELCOPY's return code.

MOD   CHA_VAL   =  235L                          * Zoned decimal constant (-2,352).
```

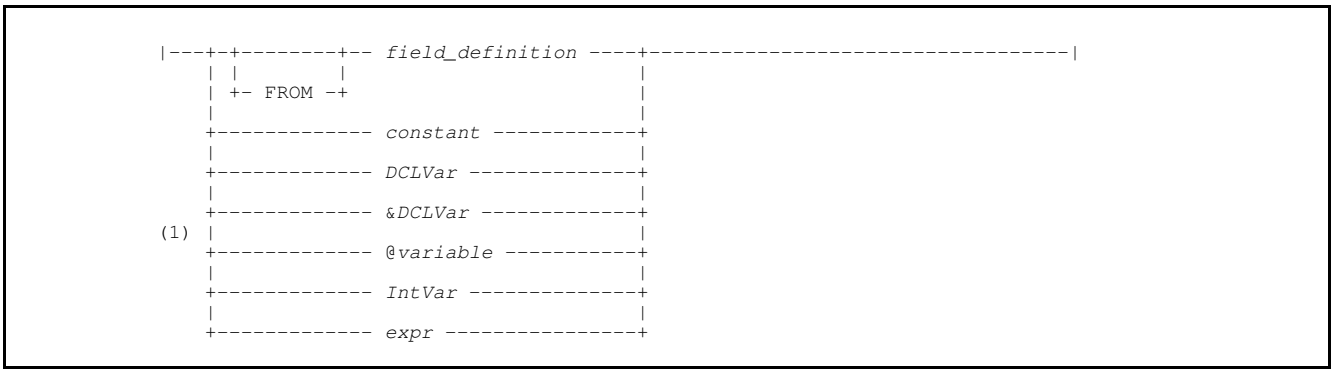
MOVE

Move (copy) a value to a declared variable or field definition.

Syntax:



Source Element:



Syntax Notes:

(1) @*variable*, *IntVar* and *expr* source elements are valid only if *target* is of numeric or numeric character data type.

Description:

The MOVE operation performs a similar function to the MOD operation in that the value represented by a source element is assigned to *target*. However, unlike MOD, only one source element value is supported.

The *source* and *target* specifications may be of different numeric or numeric character data types. If so, the MOVE operation will automatically attempt to convert the source value to the *target* data type.

For a *target* specification of character data type, the source value must also be of character data type. The source value is copied from left to right to the *target* location. If the field belonging to the *target* value overlaps the field belonging to a source value at any location beyond the first position of the source field, then the effect is as if characters are copied one at a time from the source to the *target* fields. i.e. A character that has been copied to the target field becomes a source field character that is copied later in the execution of the operation. e.g. In the following, characters "Mm." at positions 1 to 3 are copied to positions 4 to 6 respectively before characters at positions 4 to 6 are copied to positions 7 to 9, etc.

```
MOVE  'Mm.LKJNVCJ'    TO   POS 1, 18     FILL='X'   * Source: 'Mm.LKJNVCJXXXXXXXX'
MOVE  FROM   9 AT 1   TO       4                    * Target: 'Mm.Mm.Mm.Mm.XXXXXX'
```

This behaviour provides a useful technique by which character data may be propagated within a field.

If the length of a character type *target* is greater than that of the source value, then the value is padded on the right to the length of the target using the character specified by the FILL parameter. If the length of the target is less than the source value, then the value is truncated on the right.

If the *target* variable or field definition is of numeric or numeric character data type, then the source value must have a numeric interpretation. The numeric value is then converted to the data type of the target.

Parameters:

Source Element
Specifies the source element value.

If *target* is of numeric or numeric character data type, then the source element may be an @*variable*, an internal variable (*IntVar*), an arithmetic expression (*expr*) or a constant, declared variable or field definition of numeric or numeric character data type.

If *target* is of character data type, then the source element must be a constant, declared variable or field definition of character data type. Use &*DCLVar* to obtain the unformatted value of a numeric *DCLVar*.

Each source element may be specified in one of the following formats:

FROM *field_definition*
FR
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type. If *target* is of numeric or numeric character data type, then the field definition must be of numeric or numeric character data type.

If specified as a type 1 field definition but without a length, then the FROM and/or POS (field definition) keyword must be used to distinguish the *expr* that follows as a field position value instead of the source element value itself. The implied field length will be the same as *target*.

Unique to the MOVE operation, a type 3 (*field_nATp*) source element field definition may be split so that the field position specification occurs after the *target*. To do this, the following must be true:

1. The type 3 source element field definition keyword FROM must be used instead of AT.
2. If *target* is also a type 3 field definition, keyword AT must be used instead of FROM.

e.g. The following use the same type 3 character field definitions for the source element and target, and are functionally equivalent.

```
MOVE   4 AT   3      TO     4 AT   24
MOVE   4 FROM 3      TO     4 AT   24
MOVE   4             TO     4 AT   24       FROM 3
```

constant
A constant representing a character, numeric or numeric character value.

If *target* is of numeric or numeric character data type, a hex character constant source value may be specified with a numeric TYPE parameter to indicate its data type. The TYPE specification matches that used for a field definition.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
Supported for numeric or numeric character *target* values only, @*variable* is the name of an @variable that has a non-null value.
If @*variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
Supported for numeric or numeric character *target* values only, *IntVar* is the name of a SELCOPY internal variable.

expr
Supported for numeric or numeric character *target* values only, *expr* is an arithmetic expression.

TO *target*
References the variable or field to which the source element value will be moved. *target* may be specified as a declared variable or a field definition of any type.

If specified as a Type 4 (*field_pFMT*) field definition, the *target* is of numeric character data type and so the source element must also be of numeric or numeric character data type.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of the source element is implied. ERROR 69 is returned if the source element also has no explicit or implicit length.

FILL *char*
PAD
Applicable only to a *target* of character data type, FILL defines the pad character (*char*) to be used when the length of *target* is less than the length of the source element value.

The pad *char* may be specified as a quoted or hex character constant of length 1. The default is the blank character.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

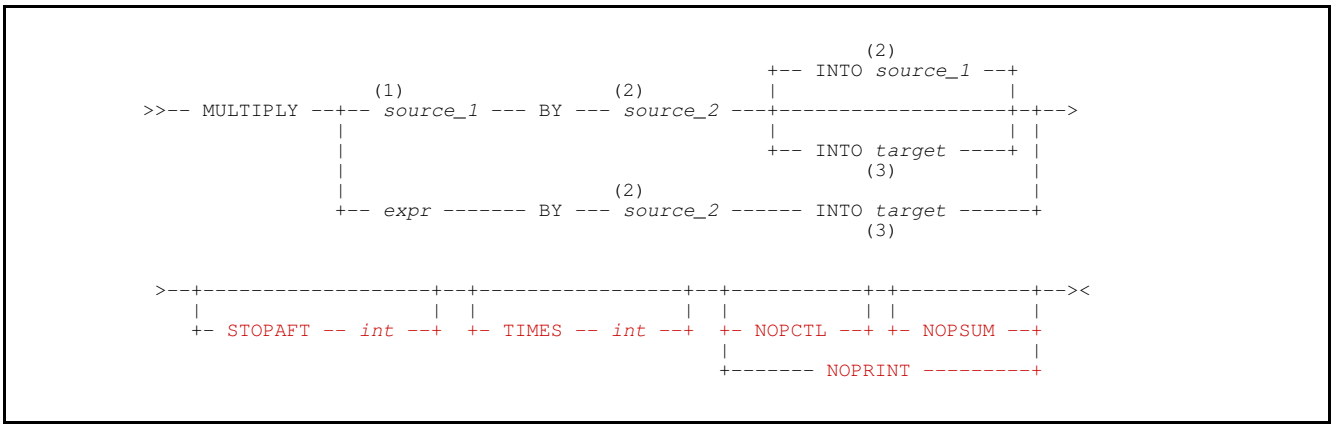
Examples:

```
DECLARE      ULINE  CHA(100)  INI='-'              * Underline "-".
DECLARE      BVAL   BIN       INI=333  FMT='SSSS9' * Binary integer.

POS  1 =  'ABC568'
POS 11 = X'0000,013C'
@y     = 55        !RC = 33            !LRECL = 245

MOVE   99 AT ULINE     TO    ULINE+1               * Extend underline to full width.

MOVE  POS  1 LENGTH=6  TO    POS 88                * Character move, target length=6.
MOVE      11, 14       TO   50 AT  91   FILL="#"   * Character padded move.
MOVE    6 AT  1        TO    3 AT  16              * Character truncated move.

MOVE       @Y          TO    5 AT  75 TYPE=P       * @variable value.
MOVE    4 AT 11 TYPE=P TO    4 AT 436 TYPE=B       * Same as CVPB operation.

MOVE      BVAL         TO    2 AT 155 TYPE=P       * Binary->Packed Decimal field.
MOVE    @y+LRECL-RC    TO    BVAL                  * Same as BVAL = @y+LRECL-RC (267).

MOVE    @y+LRECL-RC  @y+LRECL-RC+5   TO   301      * Source is type 2 field definition (field_p1p2).

MOVE      '62'         TO         301 FMT='9999'   * Numeric character source and target.
MOVE    3 AT  4 TYPE=Z TO         321 FMT='zzz9'   * Numeric source to character numeric target.
MOVE    4 AT 11 TYPE=B TO         326 FMT='sss9'   * Numeric source to character numeric target.
MOVE    4 AT 11 TYPE=P TO         331 FMT='9999'   * Numeric source to character numeric target.
```

MULTIPLY

Performs an arithmetic multiplication of two, signed numeric source values, the result of which is assigned to a target variable or stored in a target field.

Syntax:



Syntax Notes:

(1) ***source_1*** must be of numeric or character numeric data type and may be specified as a declared variable or as a field definition of type 2 (*field_p1p2*) or type 3 (*field_nATp*).
If INTO *target* is specified, *source_1* may also be specified as a constant value or an arithmetic expression (*expr*).

(2) ***source_2*** must be of numeric or character numeric data type and may be specified as a constant, an arithmetic expression (*expr*), a declared variable or as a field definition of type 2 (*field_p1p2*) or type 3 (*field_nATp*).

(3) ***target*** may be specified as a declared variable or as a field definition of type 2 (*field_p1p2*), type 3 (*field_nATp*) or type 4 (*field_pFMT*).

Synonyms:

MULTIPLY	MULT

Description:

Variables and field definitions that constitute the source values and target of the MULT operation may be of different numeric or character numeric data types.

When the source and target values are of different data types and/or if an arithmetic expression or constant value is specified as a source field, then values are converted to a common numeric data type prior to performing the multiplication. The selected common data type will have a precision and scale which is sufficient to manage the maximum precision and scale of the resultant value.

Any rational number having a fraction value, will be stored internally as a double precision (8-byte) floating point number. Therefore, all fields in the operation will be converted to double precision floating point data type and then floating point arithmetic used.

If a target or source value is expressed as a field definition without a TYPE parameter, the default data type attributed to that field will be the same as the first field definition within the command syntax to have TYPE specified. If no such field definition exists, the default data type is packed decimal integer (TYPE=P). e.g.

```
MULT  4 AT 1 TYPE=C  BY  4 AT 6 TYPE=B   INTO 4 AT 11          * Default TYPE=C
MULT  4 AT 1 TYPE=B  BY  4 AT 6 TYPE=C   INTO 4 AT 11          * Default TYPE=B
MULT  4 AT 1         BY  4 AT 6          INTO 4 AT 11 TYPE=C   * Default TYPE=C
MULT  4 AT 1         BY  4 AT 6          INTO 4 AT 11          * Default TYPE=P
```

Rounding will occur as required on the fractional part of the resultant value based on the number of fractional digits (scale) of the target definition. e.g. If the target variable or field has no scale, the result will be rounded to the nearest integer value by adding 0.5 and dropping the fractional portion.

Parameters:

source_1
Represents a rational numeric value (multiplicand) that is to be multiplied.

If no *target* is specified, then *source_1* is also the target of the operation to which the resultant value (product) will be assigned. If this is the case, *source_1* must be a declared variable or a field definition.

BY *source_2*
Represents a rational numeric value (multiplier) by which the value specified by *source_1* will be multiplied.

expr
Represents a numeric or character numeric constant, or an arithmetic expression to be used as the multiplicand value.

Note that *source_2* may also be specified in this way. The distinction is made for *expr* and *source_1* because *source_1* must also be a valid target if INTO is not specified. Therefore, if *expr* is used, INTO *target* is mandatory.

INTO *target*
Identifies the declared variable or field definition that is the target of the operation and to which the resultant value (product) is assigned. If the first source value is *expr*, then a *target* is mandatory.

If required, data conversion and decimal rounding will be performed on the resultant value before it is assigned to *target*.

Default is *source_1*.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Examples:

```
  DECLARE  PI      FLOAT   INI=3.1416          * Declared variable.
  DECLARE  AREA    FLOAT                       * Declared variable.

  EQU      OREC    1001                        * Output record area.

  OPTION   WORKLEN 2000                        * Work area buffer.

  @RADIUS     =    4 AT  11 TYPE=B             * Assign @variable value.
  AREA        =    PI
  MULT AREA             BY  @RADIUS    TIMES=2  * Area of a circle.

  MULT 4 AT 11 TYPE=B   BY  8 AT  23 TYPE=P      INTO 10 AT 101 TYPE=Z
  MULT 16.431           BY     20               INTO  8 AT 161 TYPE=F BIN
  MULT AREA             BY  8 AT 161 TYPE=F BIN  INTO  OREC+27   FMT='S,SS9.99'
```

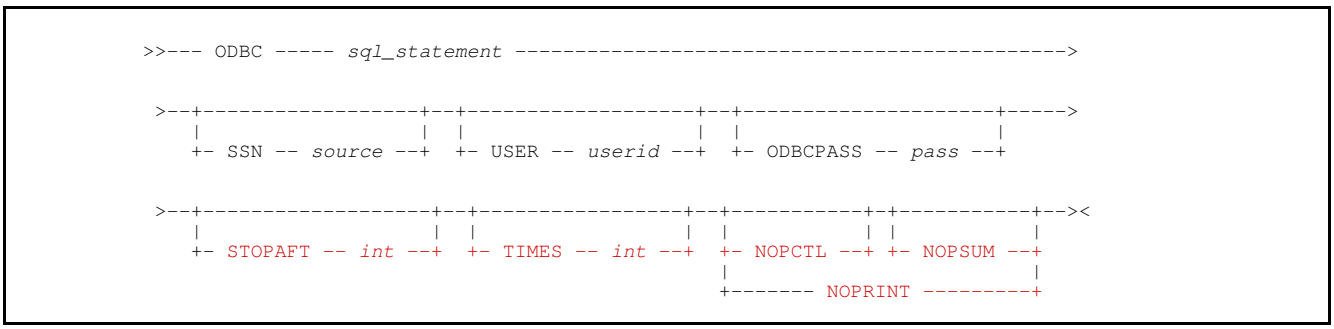
Return Codes:

0	Successful completion.
8	One of the following conditions has occurred: 1. *target* is of data type binary or floating point, and arithmetic overflow has occurred. i.e. Multiplication of 2 positive values or 2 negative values returns a negative result or multiplication of a positive value and a negative value returns a positive result. 2. The precision of *target* is not sufficient to contain the resultant value. Truncation has occurred with the loss of significant digits. 3. The first byte of a source or target field definition is within the work area but the last byte is located beyond the end of the work area buffer. The length of the field is reduced so that the last byte of the field is the last byte of the work area buffer. 4. At least one source value is treated as being of packed decimal data type but the source data is invalid packed decimal data.

ODBC

Execute an SQL statement for an ODBC connection.

Syntax:



Description:

Currently supported only on Microsoft Windows platforms, the ODBC operation will execute an SQL statement in the ODBC connected database. Selection processing will pause until control is passed back to the SELCOPY program following completion of the statement execution.

Database table input is supported for a variety of proprietary database products using ODBC (Open Database Connectivity).

ODBC is an API (Application Programming Interface) that SELCOPY uses to access most of the popular databases supporting SQL (Structured Query Language). Each DBMS (DataBase Management System) has its own driver which translates the SQL statements as required.

Specification of a system name, ODBC userid and password is mandatory to identify the ODBC Data Source Name for the required database and to establish the user's access authority to the database tables. These may be specified on the individual ODBC operation or as environment options via the OPTION operation.

Parameters:

sql_statement
Specifies the SQL statement to be executed.

The *select_clause* value may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If *sql_statement* is a quoted character constant, a default of STOPAFT 1 is implied. This may be overridden by specifying an explicit STOPAFT value.

For details on valid SQL statement syntax, please refer to SQL reference documentation for the DBMS to which the ODBC connection will be made.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

ODBCPASS *pass*
OPASS
PASSWORD
PASSWD
PASS
Specifies the password (*pass*) to be used, together with the associated user name (see USER parameter), when connecting to the data source via ODBC. Note that a user name and password is mandatory for connection to any data source.

pass must be specified as a character constant and its specification will override the prevailing value set by the ODBCPASS environment option.

SSN *source*
Specifies the ODBC data source name (*source*) to which the ODBC connection will be made. Note that specification of an ODBC source name that has been defined to the local system, is mandatory for ODBC table input.

source must be specified as a character constant and its specification will override the prevailing value set by the SSN environment option.

STOPAFT *int*
If *sql_statement* is a quoted character constant, a default of STOPAFT 1 is implied. See common parameter STOPAFT for details.

USER *userid*
Specifies the user name (*userid*) to be used, together with the associated password (see ODBCPASS parameter), when connecting to the data source via ODBC. Note that a user name and password is mandatory for connection to any data source.

userid must be specified as a character constant and its specification will override the prevailing value set by the USER environment option.

TIMES *int*
See common parameter TIMES for details.

Example:

The following sample output is from a SELCOPY program that creates a table, populates the table rows, reads the rows back then finally drops the table.

The example demonstrates use of the ODBC operation to execute a SQL CREATE, COMMIT, INSERT and DROP statements.

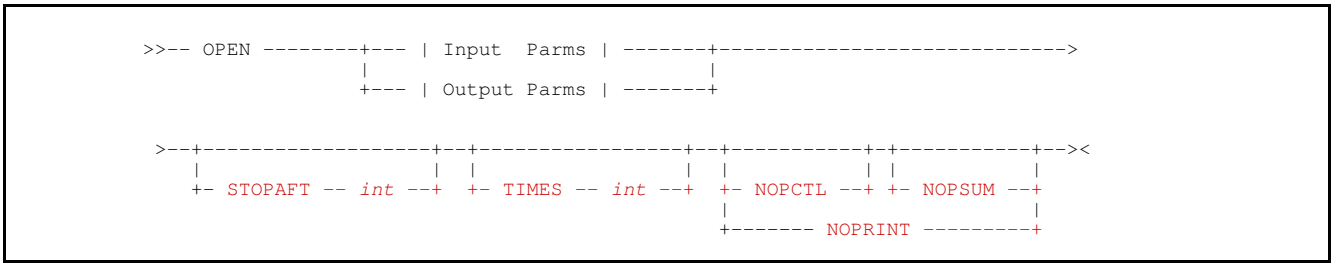
```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2016/01/07 15:21   PAGE   1
-------------------------------------------------------            ----------------   --------


          option  cblsqlog='c:\tmp\ssdb2m16.log'  user=NBJ  odbcpass=NBJ  ssn=CBLAZOS
          option  datawidth=80  worklen=2048

                          ** Table Create **

    1.    odbc  "create table nbj.ssidemo                                        \
                        (                                                        \
                         char     char(20)    not null with default,             \
                         int      int         not null with default,             \
                         date     date        not null with default,             \
                         time     time        not null with default              \
                        )"
    2.    odbc  "commit"

                          ** Table Row Insert **
    3.    odbc  "insert into  nbj.ssidemo values ('First row ',1,current date,current time)"
    4.    odbc  "insert into  nbj.ssidemo values ('Second row',2,current date,current time)"
    5.    odbc  "insert into  nbj.ssidemo values ('Third row ',3,current date,current time)"
    6.    odbc  "commit"


        ==readloop==       ** Table Row Print **
          --------
    7.    read   ssidemo   sql="select * from nbj.ssidemo"

          if eof ssidemo
    8.       then goto drop

    9.    print
   10.    goto readloop


        ==drop==           ** Table Drop **
          ----
   11.    odbc  "drop   table nbj.ssidemo"
   12.    odbc  "commit"


   INPUT   SEL SEL                                                                  RECORD
   RECNO   TOT ID.          1         2         3         4         5         6         7         8  LENGTH
   -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
       1     1   9 First row           |1          |2016-01-07|15:21:49|                              53
       2     2   9 Second row          |2          |2016-01-07|15:21:49|                              53
       3     3   9 Third row           |3          |2016-01-07|15:21:49|                              53
                   ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1             1 ODBC                                          create table nbj.ssidemo ( char char(20) not null with de ...
    2             1 ODBC                                          commit
    3             1 ODBC                                          insert into nbj.ssidemo values ('First row ',1,current da ...
    4             1 ODBC                                          insert into nbj.ssidemo values ('Second row',2,current da ...
    5             1 ODBC                                          insert into nbj.ssidemo values ('Third row ',3,current da ...
    6             1 ODBC                                          commit

                    =readloop=
    7             3 READ SSIDEMO    2048    53 F              3  select * from nbj.ssidemo
    8             1
    9---10        3

                    =drop=
   11             1 ODBC                                          drop table nbj.ssidemo
   12             1 ODBC                                          commit


    ** SELCOPY/WNT 3.30.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 12. ODBC SQL statement execution.

OPEN

Explicitly open a data object for input or output.

Syntax:



Description:

The OPEN operation will open the data object. If a specific (or implied) *fname* assigned via a READ (or CAT) operation matches that assigned to an OPEN operation which follows, then the open will be for input. Otherwise, the data object is opened for output.

By default, open of a data object referenced by a variable name or field definition on an OPEN, READ or WRITE operation, will be deferred until the operation is executed at selection time. Otherwise, in the absence of an explicit OPEN, a data object referenced by a character constant will be opened before selection time processing. An input data object is opened when the control statement containing the READ operation is processed during control statement analysis. An output data object is opened once all control statements have been analysed.

Regardless of the method used to reference the data object, the OPEN operation may be used to defer the open of an input or output object until the READ, WRITE or OPEN is executed during selection time processing.

Since open for input occurs immediately on statement analysis of a READ operation, then, in order to defer the open, either the DEFER parameter should be specified on the READ or the OPEN operation for the same *fname* must occur before the READ operation.

If the object referenced by *fname* is already open when the OPEN operation is executed, then the it will be closed and then re-opened. This may be desirable if data is to be re-processed sequentially from the start of the object. (See example 1. below) Other instances where OPEN may be useful include:

- To reuse a file name (*fname*) for multiple input or output data objects referenced by a variable name or field definition.
- To write an empty file (e.g. z/OS VSAM data set) for a file object with *fileid* referenced by a variable name or field definition. Once opened for output using the OPEN operation, the file can be closed without ever having been written to. This will clear the file data and write an end-of-file marker.

Parameters specified on the OPEN operation are those supported by a READ operation for open or data retrieval, or those supported by a WRITE operation for open or data output. Unless overridden by a subsequent READ or WRITE operation on *fname*, these parameters apply to all subsequent I/O operations on *fname*.

Parameters:

`Input Parms`
In addition to a specified or implicit *fname*, OPEN for an input data object supports any of the **Object Open** or **Data Retrieval** parameters supported by the READ operation as follow.

fileid	EOL	RAW	SSN
BDW / NOBDW	ESDS	RDW / NORDW	SUBDIR
BLKSIZE	FILL	RECFM	TABS
DEFER	FMT	RRDS	TABLE
DIR	HEADER	SELECT	UPD
DIRDATA	KSDS	SEP	USER
DIRTYPE	LIST	SORT	VSAM
DSN	LRECL	SORTDIR	WHERE
DSNPFX	ODBCPASS	SQL	

See operation READ for details on parameter usage and descriptions.

`Output Parms`
In addition to a specified or implicit *fname*, OPEN for an output data object supports any of the **Object Open** or **Data Output** parameters supported by the WRITE operation as follow.

fileid	EOL	LRECL	TABLE
APPEND	ESDS	ODBCPASS	TRUNC / NOTRUNC
BDW / NOBDW	FMT	RDW / NORDW	USER
BLKSIZE	FILL	RECFM	VSAM
DEFER	KEYLEN	REUSE	WIN
DSN	KEYPOS	RRDS	
DSNPFX	KSDS	SSN	

See operation WRITE for details on parameter usage and descriptions.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of OPEN for the same data object is unnecessary and a waste of resource, therefore, TIMES should never be used on OPEN. See common parameter TIMES for details.

Example:

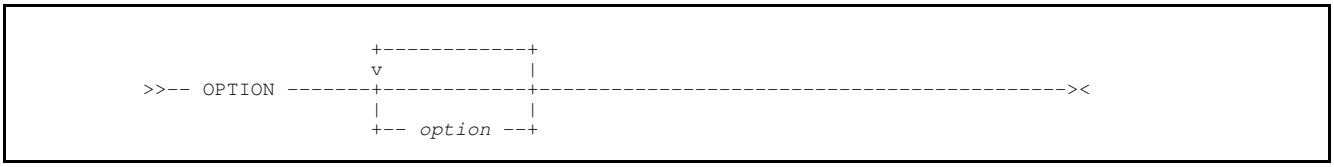
In the following example, records belonging to each member of a job library are read and compared against those belonging to a single job member master copy. The master copy is re-opened for each new member in the job library in order to start the data comparison from the first record. This processing proceeds until a matching job member is found or all members in the library have been compared.

```
EQU YES 1                                    * Flag setting on.
EQU NO  0                                    * Flag setting off.

@MATCH              = NO    STOPAFT=1        * Initialise match flag to "No".

READ INDIR  DSN='NBJ.JCL(*)'   DIRDATA   WORKLEN=1000   * JCL members.

IF  DIR                                                  * Library directory.
  THEN IF @MATCH     = YES                               * All records match?
     THEN PRINT '1ST MATCHING MEMBER: '  FROM 8 AT 901  * Print member name.
     THEN GOTO EOJ                                       * End-of-job.

     ELSE MOVE 8 AT 1   TO    901                        * Next member name.
     THEN OPEN INMAST DSN='NBJ.MASTER.JCL(BL02)'         * Re-open master.
     THEN GOTO GET                                       * Get the data record.

READ         INMAST INTO 501                 * Read next master record.

IF EOF       INMAST                          * If master end-of-file...
OR  POS 1          POS 501   LENGTH=LRECL    * ... or records don't match.
   THEN @MATCH       = NO                    * Set match flag to "No".
   THEN FLAG EOM                             * Flag end-of-member.
   ELSE @MATCH       = YES                   * Otherwise, match flag is "Yes".
```

OPTION

Set environment options that define characteristics of the SELCOPY program execution.

Syntax:



Synonyms:

OPTION	OPT OPTIONS

Description:

The OPTION operation sets the value of one or more SELCOPY environment options and is classed as one of the control statement analysis operations.

Unless documented otherwise, the prevailing value of an individual environment option may be set more than once during control statement analysis. This may occur as a result of analysis processing of another OPTION, READ, WRITE, PRINT or LOG operation which also specifies a value for the option parameter (e.g. PAGEDEPTH). However, on completion of control statement analysis processing, the prevailing value of an option will remain constant throughout selection time processing and until end-of-job.

Most environment options have a default setting. However, with the exception of those options that can only be specified within program control statements, option values may be set globally so that they become the default for all SELCOPY programs. This is achieved by specifying one or more OPTION operations in the SELCNAM options file.

Options that specify unique SELCOPY licensing details, must be included in the SELCNAM file for successful operation of a SELCOPY program. Specification of any of these options in program control statements is invalid and will return a control analysis error. These options are as follow.

PASS	RANGE	SITE	

Similarly, the following environment options are invalid when specified in the SELCNAM file and so are only supported within the program's control statements.

DSNPFX	HEAD	REPORT	WORKLEN

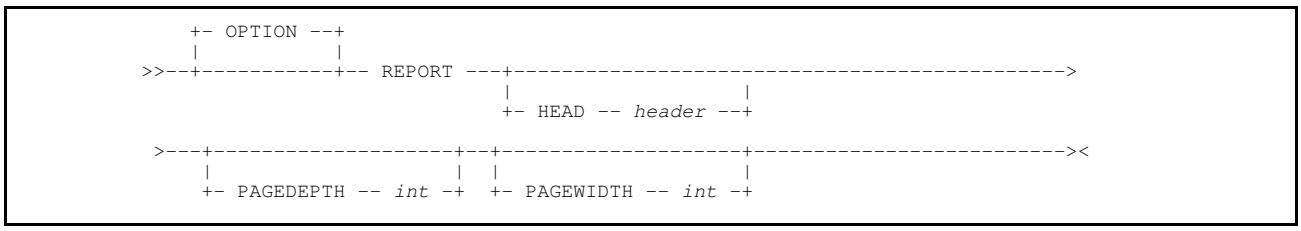
The following options may be specified in both the SELCNAM file and program control statements.

ABTRAP	DUMPALL	NOPRINT	RC_KEYNF
BANNER	DUMPENC	ODBCPASS	RDW
CALLTYPE	ENVFAIL	PAGEDEPTH	SEP
CBLSQLOG	ENVVAR	PAGEWIDTH	SORTDIR
DATAWIDTH	ERRLIM	PRINTABLE	SSN
DEFAULTFP	FILL	PRRECLEN	SUBDIR
DEFDIR	LIBNAME	PRTCTL	UNPRINTABLE
DIRTYPE	MFC	PRTSUM	USER

Use of the OPTION operation keyword is optional in the following special cases:

Report Output

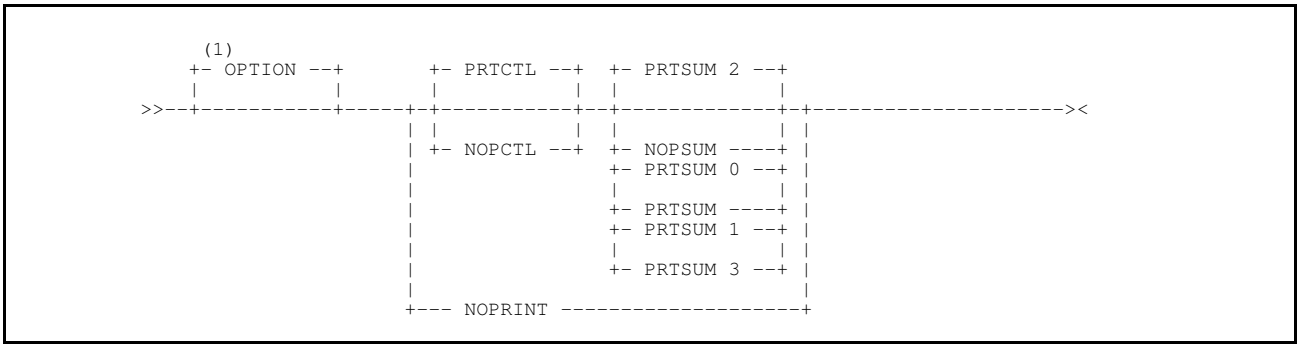
SELCOPY report output is triggered by the presence of option REPORT which, if the first value specified on the OPTION operation, may be specified with or without the OPTION keyword. Other options relating to report output may be specified following the REPORT option.



SELCOPY List Diagnostic Output

By default, SELCOPY list output includes diagnostic output which includes print of the input control statements with their assigned selection id numbers, and also a summary block containing an execution count for each selection.

Display of this diagnostic output may be switched off by options set in the SELCNAM file, but may be re-activated and deactivated within the program control statements. When specified in the program statements, these options may be specified with or without the OPTION keyword.



Syntax Notes:

(1) Keyword OPTION is mandatory if option PRTSUM is specified.

Parameters:

ABTRAP **ON** | **OFF**
TRAP **ON** | **OFF**
NOTRAP

Controls whether SELCOPY's default abnormal end interrupt handling is enabled or disabled.

If ABTRAP is set ON, SELCOPY will intercept any abnormal end situation and terminate the job quietly, suppressing the default system action of producing diagnostic information. For Windows and POSIX environments, this includes interruption of the program execution by the user.

Unless unset by the SELCOPY command option -notrap , the default setting for ABTRAP is ON.

BANNER | **NOBANNER**
BAN | **NOBAN**

Controls whether or not the SELCOPY banner is written to log output (STDERR or SYSOUT) when SELCOPY is started. e.g.

```
SELCOPY/MVS 3.30 at CBL - Bridgend UK (Internal Only)        2015/11/17 16:19
```

The SELCOPY banner line is not written until control statement analysis is completed and so the last setting for BANNER or NOBANNER will be obeyed. However, if NOBANNER is set and an error is detected during control statement analysis, option BANNER is set and the banner line is logged before the error message.

Default is BANNER.

CALLTYPE **DIRECT** | **VIA_SLCCALL**

Applicable only to CALL operations on Windows and Unix systems. CALLTYPE controls the type of linkage used by all routines executed via CALL operations in the SELCOPY program. The options are as follow:

DIRECT

This the recommended option. Routines in the run-time shared library will be called directly with a single argument which is a pointer to an array of character string pointers. i.e. `(char **list)`

The name of the shared library is the prevailing value for the LIBNAME environment option.

VIA_SLCCALL

This option should be used for legacy (pre version 2.09) SELCOPY jobs. Routines are called from the user-written function, **slccall**, which exists in the a shared library named as follows:

- **slccall.dll** for Microsoft Windows.
- **slccall.so** for most UNIX.
- **libselc.sl** for HP Unix.

The linkage for the slccall function is: `(char *nam, char **list)`
The first argument is a pointer to the null terminated name of the routine to be called, and the second is a pointer to an array of character string pointers.

The slccall function must then call the named routine passing the parameter list array and/or any other arguments required by the routine linkage.

When the first CALL operation is processed during control statement analysis, the run-time shared library with the library name determined by the CALLTYPE and, if applicable, LIBNAME options, is dynamically loaded and thereafter used as the source for all routines executed by CALL operations. i.e. SELCOPY supports dynamic load of only one run-time shared library during program execution.

Although not recommended, it is possible to change the setting for CALLTYPE so that different linkage is used on CALL operations within the same SELCOPY program. Note, that regardless of the linkage used, the shared library that was dynamically loaded during control statement analysis, will be used for lookup of all called routines.

The shared library must exist in a directory included in the system library path. The library path is identified by system environment variable PATH in Microsoft Windows, LIBPATH in IBM AIX, and LD_LIBRARY_PATH in most other Unix environments.

The default CALLTYPE option is DIRECT.

CBLSQLOG *fileid*
LOGSQL
SQLLOG

Applicable to ODBC table input and output, CBLSQLOG indicates to SELCOPY that a log file is to be created containing a record of calls made to the ODBC Driver Manager.

This may be particularly useful since SQL error messages can sometimes be very long, resulting in truncation when displayed in the SELCOPY list output. However, no truncation occurs when written to the CBLSQLOG file or when the message is written to the terminal at execution time.

The CBLSQLOG option may also include specification of *fileid*, the fileid path of a new or existing file to which the SELCOPY ODBC log will be written. *fileid* must be quoted character constant identifying a Microsoft Windows fileid clause specification.

If CBLSQLOG is specified without *fileid*, the default is "selcSQL.log".

DATAWIDTH *int*
DW

Applicable only to PRINT and LOG operations where the format applied to the output data is not TYPE S (system) or TYPE D/DX (dump), DATAWIDTH specifies the width of the printed data column in the print block. i.e. the length of printed data to be written to each line of the SELCOPY list output.

In addition to the OPTION operation, a value for DATAWIDTH may be specified on a LOG, PLOG or PRINT operation. Therefore, the actual value assigned to the DATAWIDTH option is established by the last OPTION, LOG, PRINT or PLOG operation within the control statements on which DATAWIDTH is specified. All other DATAWIDTH value specifications are overridden by this last specification.

This DATAWIDTH value will apply to all PRINT and LOG operations executed during the SELCOPY program run.

The maximum value is 4096, the minimum value is 10 and the default value is 100.

DEFAULTFP **BIN** | **HEX** | **NAT**
DEFFP
DFLTFP

DEFAULTFP specifies the floating point style to be used when interpreting floating point field definitions or declared variables for which no style has been specified. e.g. for a CVFx or CVxF operation.

The options are as follow:

BIN	IEEE-754 Base 2 Binary style.
HEX	IBM Base 16 Hexadecimal style.
NAT	Native style (BIN or HEX) for the machine architecture. On z/OS and z/VM CMS, the native floating point style is HEX. Otherwise, the native style is BIN.

Beware that the native floating point style is used by SELCOPY for all of its floating point arithmetic and conversion. Use of the non-native style will involve data type conversion and cost CPU time.

Default is NAT.

DEFDIR *path*
DEFAULTDIR
DFLTDIR

Applicable only to Windows and POSIX environments, DEFDIR identifies *path* as the work (current) directory to be used by SELCOPY for the duration of the program execution only. Note that no change is made to the system's current working directory.

The *path* value may be an absolute directory path or one relative to the system's current working directory, and must be specified as a quoted character constant or an unquoted literal. Character case will be preserved even if unquoted, however, beware of ending an unquoted *path* with "\" (backslash) which will be treated as the statement continuation character if the last non-blank character on the input control record.

path may also include references to environment variable and/or parameter variable substitution symbols. However, if using an equated symbol for *path*, the equated value should be specified as a quoted character constant in order to maintain character case. e.g.

```
** Windows Example:
  opt    defdir 'c:\Tmp'
  read   abcfile.txt        * Input  file:  "c:\Tmp\abcfile.txt"

** UNIX    Example:  (System current working directory is  "/home/NBJ" )
  opt    defdir Selc/out     * Mixed case will be respected.
  write  test/defdir.txt     * Output file:  "/home/NBJ/Selc/out/test/defdir.txt"
```

The default work directory is used to establish the location of a fileid specified without a path or with a path relative to the work directory. This fileid may be specified on an I/O or INCLUDE operation.

The DEFDIR option must not be set on a statement following one containing an I/O operation. If so, ERROR 169 will be returned on the I/O operation regardless of whether the fileid of the input or output file references an absolute file path.

Default is the system's current working directory at the time the SELCOPY program was started.

DIRTYPE *attributes*
TYPEDIR

Applicable to directory record input via the READ DIR or READ DIRDATA operation for Windows and Unix-like file directories only.

The DIRTYPE option specifies the types of directory entries to be selected for processing based on the attributes of the files they reference. Attribute codes are as follow:

Code	File Attribute required
.	The working "." and parent ".." directory entries.
R	Read-only.
H	Hidden.
S	System.
V	Volume-Id.
D	Subdirectory.
A	Archive. (Amended file.)
Y	Symbolic Link. (UNIX only.)
N	Normal. (No attribute flags set.)

Option keyword **ALL** may be specified to indicate that all directory entries are to be included.

Unless ALL is specified, multiple DIRTYPE attribute codes may be specified as an unquoted literal or as a quoted character constant. Blanks and comma characters may be specified and are ignored if supplied as a quoted character constant.

The default DIRTYPE attribute values are 'A,N,R' which may be overridden by a DIRTYPE specification on a READ operation.

DSNPFX **YES** | **NO**
USERID
UID

Applicable only to z/OS where an INCLUDE or I/O operation references an unquoted fileid which is a data set name (DSN) belonging to a PDS/PDSE library or a sequential or VSAM data set. The fileid reference may have been specified as an unquoted literal or an unquoted value identified by a declared variable or field definition.

Valid only in program control statements, OPTION DSNPFX determines whether an attempt will be made to open the data set using one of the following prefixes applied to the unquoted data set name:

- ◊ The ACF userid assigned to the job if running in batch.
- ◊ The current TSO prefix definition if running in a TSO/E environment.

On a z/OS system, SELCOPY attempts to open a file using a fileid which is established in the following order of precedence. If the open fails for that fileid, then, unless otherwise specified, SELCOPY will re-attempt an open using a fileid established by the next definition in the sequence order.

1. Whether or not it is quoted, if the specified fileid is not a valid z/OS DSN, it is treated as a Unix-like HFS/ZFS fileid. If the open fails for this file, no further attempt will be made to open a file using a different fileid.

2. If the fileid is quoted, it is treated as an actual DSN with no prefix applied.

3. If the fileid is unquoted, the DSNPFX option value takes effect as follows:

    1. If **DSNPFX YES** is set, the DSN prefix will be applied to fileid. If the open fails, then unless the I/O operation is READ with an explicit DSNPFX YES parameter specified, the open is re-attempted using option DSNPFX NO.

    2. If **DSNPFX NO** is set, no DSN prefix will be applied to fileid.

4. The fileid is treated as a Unix-like HFS/ZFS fileid.

The default is DSNPFX YES which may be overridden by a DSNPFX NO (or NODSNPFX) specification on an I/O operation.

DUMPALL **YES** | **NO**

Applicable to the LOG, PLOG and PRINT operations that output data in dump (TYPE D or DX) format.

For a group of consecutive lines of dump format output that contain identical data, DUMPALL determines whether or not the second and subsequent lines of the group are to be condensed. If so, these lines are displayed as a single line containing the literal "=same=", followed by a parenthesised count of the number of condensed lines. e.g.

```
   INPUT   SEL SEL
   RECNO   TOT ID.
   -----   --- ---
       0     1  11           190
0000  206C696E 65732073 75707072 65737365     | lines suppresse|
0010  642E2020 20202020 20202020 20202020     |d.              |
0020  20202020 20202020 20202020 20202020     |                |
0030       =same=          (7 lines)          :                :
00A0  20202020 20202020 44696666 20646174     |        Diff dat|
00B0  612E2020 20202020 20202020 2020         |a.              |
```

DUMPALL YES will output all line containing duplicate data, DUMPALL NO will condense these lines. Note that DUMPALL is a synonym for DUMPALL YES.

The default is DUMPALL NO which may be overridden by a DUMPALL specification on a PRINT operation.

DUMPENC *char_string*

Applicable to the LOG, PLOG and PRINT operations that output data in a dump format which includes character representation of the data. (i.e. TYPE D)

DUMPENC specifies *char_string*, a character constant of length 1 or 2, which identifies the characters used to enclose the character representation of dump format output.

1. The first (or only) character defines the enclosing character used on print lines that have not been condensed to a single line. (See option DUMPALL above).

2. The second character defines the enclosing character used on print lines that have been condensed to a single line.

The default DUMPENC characters are "|" (or symbol) and ":" (colon) which may be overridden by a DUMPENC specification on a PRINT operation.

ENVFAIL *char_string* | **CANCEL** | **NULL** | **SAME**

If option ENVVAR is set, then ENVFAIL defines the action taken if a control statement makes reference to an environment or parameter substitution variable that has not been assigned.

The possible options are:

char_string	A character constant that will be substituted in place of the variable name.
CANCEL	Cancel the execution. (See the GOTO CANCEL operation.)
NULL	A null string will be substituted in place of the variable name.
SAME	The variable name will not be substituted but will remain unchanged in the control statements.

If required, option ENVFAIL may be specified with different values throughout the input control statements to change the action taken for failed variable substitution in the statements that follow.

Default is ENVFAIL SAME.

ENVVAR | **NOENVVAR**

Controls whether or not variable substitution of environment and parameter variables is activated.

If required, option ENVVAR and NOENVVAR may be specified throughout the input control statements to respectively activate and deactivate variable substitution for the statements that follow.

The default setting is ENVVAR, i.e. substitution is activated.

ERRLIM *int*
ERRMAX

ERRLIM specifies *int*, a non-zero integer numeric constant that identifies the number of errors that may be detected and reported during control statement analysis before the run is terminated.

Once this error limit threshold has been reached no further control statement analysis takes place and, therefore, any errors that exist in subsequent, unprocessed statements will not be detected.

The default value for ERRLIM is 10.

FILL *char*
PAD

The FILL option specifies *char*, a character constant of length 1. The FILL option may be set within the program control statements and the SELCNAM file where they have different interpretations.

- ◊ If set in the program control statements, then all characters in a defined work area (option WORKLEN) will be initialised as *char*.
- ◊ If set in the SELCNAM file, then *char* will be used to pad a short input record to the length of the last record previous read from the same data object. (See parameter FILL on the READ operation for more detail).

The default FILL character is blank in both cases.

HEAD *header* | **NO**
HD
H

Valid only in program control statements, OPTION HEAD specifies *header*, a quoted character constant which identifies the header string to be displayed in the header line of each page in the SELCOPY list output that follows.

The *header* value will replace the *company_name* as specified by the option SITE in the SELCNAM file. The maximum length of the *header* text displayed in the header line is 56 characters. If a *header* value longer than 56 characters is set, then it is truncated without warning.

HEAD NO may be set if the header text within the header line is to be suppressed. Note that this does not affect display of the time, date and page number.

The HEAD option is one which may be specified following option REPORT without the OPTION operation keyword. The *header* text may be referenced and, if necessary, updated during selection time processing using the internal field definition located at position HEAD.

LIBNAME *library*

Applicable only to CALL operations on Windows and Unix systems where environment option CALLTYPE is set to DIRECT. LIBNAME identifies *library*, a quoted character constant which identifies the name of the shared library in which all routines referenced by a CALL operation may be found.

Unless the default is to be used, the LIBNAME option should set before any statements containing a CALL operation. When the first CALL operation is processed during control statement analysis, the shared library with a name equal to the prevailing value for *library*, is dynamically loaded and then used as the source for all routines executed by subsequent CALL operations. Any change to the LIBNAME *library* following processing of the first CALL operation, will have no effect.

The shared library must exist in a directory included in the system library path. The library path is identified by system environment variable PATH in Microsoft Windows, LIBPATH in IBM AIX, and LD_LIBRARY_PATH in most other Unix environments.

The default value for *library* is **libselc.dll** for Microsoft Windows and **libselc.so** for Unix systems.

MFC

Indicates that VSAM related parameter keywords on I/O operations are to imply use of the Micro Focus file handler. This option is ignored on z/OS and z/VM CMS where IBM VSAM is used.

NOPRINT
NOP

Equivalent to specifying both environment options NOPCTL and NOPSUM. (See PRTCTL and PRTSUM respectively)

If specified as the only option value, NOPRINT may be specified without the OPTION keyword. Furthermore, NOPRINT is a common parameter which may be specified on any operation or assignment statement.

ODBCPASS *pass*
OPASS

Applicable to ODBC table input and output, ODBCPASS specifies the password (*pass*) used to connect to the data source via ODBC. *pass* must be specified as a character constant.

The *pass* value may be overridden by specification of the ODBCPASS parameter on a READ or WRITE operation.

PAGEDEPTH *int*
PD

Specifies the depth (number of lines) of each page of the SELCOPY list output.

When a line written to the SELCOPY list output exceeds the PAGEDEPTH threshold, then, unless the prevailing value for environment option HEAD is NO, the page headers will be written at the start of the new page. The exception is when a PRINT TYPE=S (system print) operation is executed. For system print output, the PAGEDEPTH value is ignored and output to a new page is only achieved by specifically including ASA character 1 in the first position of the printed text.

In addition to the OPTION operation, a value for PAGEDEPTH may be specified on a LOG, PLOG or PRINT operation. Therefore, the actual value assigned to the PAGEDEPTH option is established by the last OPTION, LOG, PRINT or PLOG operation within the control statements on which PAGEDEPTH is specified. All other PAGEDEPTH value specifications are overridden by this last specification.

However, while control statements are being analysed, SELCOPY writes lines to the list output. Therefore, during control statement analysis only, the depth of each page is volatile and is determined by the prevailing PAGEDEPTH value as established by the control statements analysed thus far. i.e. The page depth threshold for the current output page will change when another PAGEDEPTH parameter is processed by an, as yet unanalysed, control statement.

The maximum value is 2,147,483,647 (X'7FFF,FFFF'), the minimum value is 10 and the default value is 58.

PAGEWIDTH *int*
PW

Specifies the width of each page of the SELCOPY list output.

In actual fact, the PAGEWIDTH value affects only those lines of the list output that contain page headers and dump format print output (TYPE D/DX). All other lines of the list output (control statements, print block data, summary block data, warning messages and footer text) are unrestricted by the PAGEWIDTH value.

Within each page header, the current date, time and page number, are right adjusted at the PAGEWIDTH value. If the PAGEWIDTH value is sufficiently small, then these header items will overlap and so overwrite the title, which is left adjusted within the header line.

For PRINT or LOG operation dump format output, the length of the output line and, therefore, the length of printed or logged data represented by that line, is restricted by the PAGEWIDTH value. See description of TYPE=D in the print block output for details on how PAGEWIDTH affects dump format print.

In addition to the OPTION operation, a value for PAGEWIDTH may be specified on a LOG, PLOG or PRINT operation. Therefore, the actual value assigned to the PAGEWIDTH option is established by the last OPTION, LOG, PRINT or PLOG operation within the control statements on which PAGEWIDTH is specified. All other PAGEWIDTH value specifications are overridden by this last specification.

However, while control statements are being analysed, SELCOPY writes lines to the list output. Therefore, unless option NOPRINT or NOPCTL is set on the first control statement, at least one page header will be written before selection time processing starts. The length of each page header is determined by the prevailing PAGEWIDTH value established by the control statements analysed thus far. i.e. PAGEWIDTH values set by subsequent, as yet unanalysed, control statements will not affect the output header.

For z/OS and z/VM CMS only, the PAGEWIDTH value includes the column reserved for ASA print characters. Therefore, the length of page header text that would actually be printed by a physical printer device, is one less than the PAGEWIDTH value.

The maximum value is 156 and the minimum value is 66. The default value is 132 for POSIX environments and 133 for z/OS and z/VM CMS environments.

PASS *pass*

Valid in the SELCNAM file only, OPTION PASS specifies *pass*, an 8-byte hex character constant containing the SELCOPY licence key password.

Together with options SITE and RANGE, PASS is one of the licence environment options required for successful execution and is validated by SELCOPY when the program starts. The *pass* value must exactly match that provided by Compute (Bridgend) Ltd. If not, SELCOPY will terminate with ERROR 153.

PRINTABLE *char_string*

Nominates character code points that are to be treated as printable by the LOG, PLOG and PRINT operations.

The SELCOPY print/log types that involve interpretation of code points as being printable or unprintable are: character (C), dump (D) and mixed character and hex (M, MC or MP). For these print/log types, only the code points identified as being printable will be passed without translation to the output stream.

The *char_string* is a hex character constant representing one or more character points to be treated as printable. e.g. To set as printable the US EBCDIC (code page 037) characters "[" (left bracket) and "]" (right bracket) at code points x'BA' and x'BB' respectively, set `PRINTABLE X'BABB'`.
Note that, for UK EBCDIC (code page 285), "[" and "]" are at code points x'B1' and x'BB' respectively.

Default printable code points are illustrated by the description of TYPE C output for the PRINT operation.

PRRECLEN YES | **NO**
Specifies whether or not the RECORD LENGTH column is to be displayed in the SELCOPY list output Print Block.

PRTCTL | **NOPCTL**
Specifies whether or not the SELCOPY list control statement diagnostic output is active.

Output of control statements to the SELCOPY list occurs as each statement is read and processed during control statement analysis.

By default, PRTCTL is set indicating that control statement output is active. If required, options NOPCTL and PRTCTL may be positioned throughout the input control statements to suppress and re-activate output of the statements that follow.

Note that, if an error is detected during control statement analysis, PRTCTL is automatically set so that the error, and the statement to which it applies, is displayed in the list output.

If specified as the only option value other than NOPSUM, PRTCTL and NOPCTL may be specified without the OPTION keyword. Furthermore, NOPCTL is a common parameter which may be specified on any operation or assignment statement.

PRTSUM *int* | **NOPSUM**
SUMPRT *int* | **NOPTOT**
Specifies whether or not the SELCOPY list output summary (totals) block will be generated and written at end-of-job. PRTSUM (or SUMPRT) activates summary block output, whereas NOPSUM (or NOPTOT) suppresses it.

Option PRTSUM may also specify *int*, a value of 0, 1, 2 or 3, which defines the level of detail displayed in the summary block. PRTSUM 0 is equivalent to NOPSUM.

PRTSUM 1
Separate summary block entries will be generated for READ, CAT, WRITE, INSERT, UPDATE and DELETE operations and the operation keyword displayed with the data object reference information. All other operations may be grouped together if the executed count (SELTOT) is the same, and displayed on a single summary block entry. e.g.

```
SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL          FSIZE   CI    DSN
 ------      ------      ----     -------  -----          -----   --    ---
    1            22 READ SSDYN07      20    10 FB             3       C:\djh\cc\slc\SSDYN07.INP
    2             3
    3            22
    4             3
    5----6        1
    7----9        3
```

PRTSUM 2
As for PRTSUM 1, but additionally:

1. All user labels are also reported in the summary, with a preceding blank line.
2. A separate entry exists for RETURN operations. Unless a summary block comment is specified following the RETURN operation (i.e. text following *>), the RETURN keyword will be displayed as `=ret=`. e.g.

```
SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL          FSIZE   CI    DSN
 ------      ------      ----     -------  -----          -----   --    ---
                  =rdrtn=
    1            22 READ SSDYN07      20    10 FB             3       C:\djh\cc\slc\SSDYN07.INP
    2             3
    3            22 =ret=

                  =xxrtn=
    4             3
    5----6        1
    7----9        3 =ret=
```

PRTSUM 3
As for PRTSUM 2, but additionally:

1. A separate entry exists for each operation and assignment statement (including IF/AND/OR operations). Multiple operations or assignments having the same selection count are no longer grouped together and reported as a single summary block entry.
2. Comments on all operation or assignment statements are reported following the operation keyword whether or not it is a summary block comment. e.g.

```
SUMMARY..
 SEL-ID      SELTOT           FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------           ----     -------  -----           -----   --    ---
                   =rdrtn=                        * Comment data on a label.
    1            22   READ   SSDYN07        20    10 FB              3       C:\djh\cc\slc\SSDYN07.INP
    6            74   READ     FILE1      2048   128 U              75       C:\djh\cc\slc\LST\SSPARM11
    7            75   @11=                                              * Comment data on a statement.
                      if                                                 * Comment data.
    2             3     t do                                           * Comment data.
    3            22 =ret=                                              * Return from a subrtn.

                   =xxrtn=                        * Comment data on a label.
    4             3   pr                                               * Comment data on a statement.
                      if                                                 * Comment data.
    5             1     t move                                         * Comment data.
    6             1     t move                                         * Comment data.
    7             3   move                                             * Comment data.
    8             3   pr                                               * Comment data.
    9             3 =ret=                                              * Return from a subrtn.

                   =xyz_label=                    * Just a fabricated example.
                     if                                                * If something or other.
                      a
  269             0     t mod                                          * Change something.
  270             0     t rc=                                          * Set return code.
```

See description of the Summary (Totals) Block for more detail on the affects of PRTSUM.

If specified as the only option value other than NOPCTL, NOPSUM (or NOPTOT) may be specified without the OPTION keyword. Furthermore, NOPSUM is a common parameter which may be specified on any operation or assignment statement.

The default setting is PRTSUM 2. However, a specification of option PRTSUM with no *int* value will default to PRTSUM 1.

RANGE *date_range*
Valid in the SELCNAM file only, OPTION RANGE specifies *date_range*, a quoted character constant containing one of the operational date ranges for which SELCOPY is licensed.

Together with options PASS and SITE, RANGE is one of the licence environment options required for successful execution and is validated by SELCOPY when the program starts. Each specification of a RANGE option must reference a *date_range* that exactly matches that provided by Compute (Bridgend) Ltd. If not, SELCOPY will terminate with ERROR 153.

The *date_range* specification is in the form ('*yyyy*/*mm*/*dd*-*yyyy*/*mm*/*dd*'), i.e. two ISO dates separated by a "-" (hyphen/minus sign). This identifies the inclusive, start and end dates of the operational date range. If an attempt is made to execute SELCOPY outside the operational date ranges, ERROR 123 (expiry) is returned.

More than one RANGE *date_range* option may be specified to identify a number of contiguous or non-contiguous date range windows in which SELCOPY will run. Multiple *date_range* values are cumulative and must be specified if provided as part of the installation's licence key.

RC_KEYNF *int*
Applicable to direct read of input records via a READ operation with parameter KEY, KGE, RBA or REC.

RC_KEYNF specifies *int*, an integer value in the range 0 to 254 which represents the SELCOPY return code to be set when a direct read returns a record not found condition. In addition to this return code, the following 25 character record will be returned in the work area or input buffer area.

```
--- KEY/REC NOT FOUND ---
```

If RC_KEYNF 0 is set, the prevailing SELCOPY return code is unchanged and no indication of the failed read is reported against the selection identifier in the summary.

If RC_KEYNF is non-zero, the return code will be set as requested. However, for the purpose of identifying the failure in the summary report, the selection identifier will be flagged as having set return code 8 indicating a minor error condition.

Default value is 0.

RDW | **NORDW**
Applicable only to input of variable length record format (RECFM V, VB, V2, V3 and MFV) files.

RDW and NORDW indicate whether or not the Record Descriptor Word (RDW) is to be included as part of the data belonging to each input record. If option RDW is set, the length of the RDW is included in the length value assigned to variable LRECL when a record is read.

Beware that, if RDW is specified for a variable length file which is also the prime input object, then, for any WRITE operation, the length of the RDW field is added to the FROM parameter address and deducted from the output data length.

The setting may be overridden by specification of parameter RDW/NORDW on a READ operation.

For z/OS systems, the default is set by CBLNAME option SRDW= which is set to Yes (RDW) by default. Otherwise, the product default is NORDW.

REPORT
R

Valid in program control statements only, OPTION REPORT activates SELCOPY report output for printed data. Report output has the following attributes:

1. All print block headers and footers are suppressed.
2. All print block column values, other than the printed data, are suppressed.
3. On z/OS and z/VM CMS, each row of printed data starts in the second column of the list output. SELCOPY will output ASA print control characters in the first column. On all other platforms, printed data starts in the first list output column.
4. Rows of printed data do not wrap onto the next line of the list output.
5. A new page is started and the page number (referenced by internal field UXPGNO) is reset to page 1 on execution of the first PRINT operation.
6. The Control Statements and Summary (Totals) Block are automatically printed on separate pages of the list output.

See also environment options HEAD, PAGEDEPTH and PAGEWIDTH which also have an effect on the printed report output.

If specified as the first option value, REPORT may be specified without the OPTION keyword.

```
1SELCOPY/MVS 3.30 at CBL - Bridgend UK (Internal Only)              2015/11/16 15:56   PAGE   1
 ------------------------------------------------------             ----------------   --------


            report  head="Extent Usage on Volume: CBLME1"

      1.    read   inlst list='LX CBLME1'   header

            if eof inlst
      2.      then space 1

      3.    tran pos fhdr+lrecl-1, fhdr+lrecl+lrecl-1  \
                                    '.,1234567890'   '------------'  stopaft=1
      4.    print

1Extent Usage on Volume: CBLME1                                     2015/11/16 15:56   PAGE   1
 ------------------------------                                     ----------------   --------

 Vol            CC    HH Dsn                                         Org Alu      Trks Seq Nxt LoCCHH   HiCCHH
 ------ ---------- ----- ---------------------------------------------- --- -   -------- --- --- -------- --------
 CBLME1          0     0 **Label Area**                                     T          1   1   1 00000000 00000000
 CBLME1          0     1 **VTOC**                                           T        300   1   1 00000001 00140000
 CBLME1         20     1 SYS1.VVDS.VCBLME1                            VS    T         10   1   1 00140001 0014000A
 CBLME1         20    11 LAC.CBLI.DFPSORT                             PS    T          1   1   1 0014000B 0014000B
 CBLME1         20    12 LAC.SYSIN                                    PS    T          1   1   1 0014000C 0014000C
 CBLME1         20    13 **Free Extent**                                   T         63   1   1 0014000D 00190000
 CBLME1         25     1 SYS1.VTOCIX.CBLME1                           PS    T         18   1   1 00190001 001A0003
 CBLME1         26     4 **Free Extent**                                    T         11   2   1 001A0004 001A000E
 CBLME1         27     0 TEST1.EAV.PDS                                PO    C      30000   1   1 001B0000 07EA000E
 CBLME1       2027     0 LAC.LARGE1                                   PS    C     149985   1   1 07EB0000 2EF9000E
 CBLME1      12026     0 **Free Extent**                                    T     802410   3   1 2EFA0000 FFEF000E
 CBLME1      65520     0 JGE.LARGE.COPY                               PS    C     150255   1   1 FFF00000 2710001E
 CBLME1      75537     0 JGE.LARGE                                    PS    C     150255   1   1 27110010 4E31001E
 CBLME1      85554     0 TEST3.KSDS.ADA.DATA                          VS    C      45045   1   2 4E320010 59EC001E
 CBLME1      88557     0 TEST3.KSDS.ADA.INDEX                         VS    C        315   1   1 59ED0010 5A01001E
 CBLME1      88578     0 TEST3.KSDS.ADA.DATA                          VS    C       1890   2   2 5A020010 5A7F001E
 CBLME1      88704     0 **Free Extent**                                    T       5040   4   1 5A800010 5BCF001E


1Extent Usage on Volume: CBLME1                                     2015/11/16 15:56   PAGE   2
 ------------------------------                                     ----------------   --------


 SUMMARY..
  SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
  ------      ------      ----     -------  -----           -----   --    ---
     1            19 READ INLST             112 F              19       LX CBLME1
     2----3        1
     4            20


     ** SELCOPY/MVS 3.30.002  2015/09/17  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                    ** Expiry: 14 Jul 2016 **
```

Figure 13. z/OS Printed REPORT Output.

SEP *char*
SEP **NO** | **OFF**

Specifies *char*, a character constant of length 1 which represents the statement separation character used in control statements that follow. Alternatively, SEP specifies **NO** or **OFF** to disable statement separation in the control statements that follow.

Note that, if statement separation is disabled, SEP *char* will re-enable it.

For z/OS systems, the default is set by CBLNAME option, Separator=. Otherwise, the product default is "!" (exclamation mark).

SITE *company_name*
Valid in the SELCNAM file only, OPTION SITE specifies *company_name*, a quoted character constant containing the licensed company name and location.

Together with options PASS and RANGE, SITE is one of the licence environment options required for successful execution and is validated by SELCOPY when the program starts. The *company_name* specification is between 20 and 36 characters in length and must exactly match that provided by Compute (Bridgend) Ltd, respecting character case, embedded blanks and punctuation. If not, SELCOPY will terminate with ERROR 153.

By default, *company_name* is displayed in the header line of each page in the SELCOPY list output.

SORTDIR **D** | **E** | **H** | **M** | **N** | **P** | **S** | **T** | **X** | **Z** | **0** | **NO**
SORT
Applicable to directory record input via the READ DIR or READ DIRDATA operation for Windows and Unix-like file directories only.

SORTDIR may be specified to read directory entries ordered by one of the following:

Code	Sort Order
D or T	Date. i.e. The file's last-changed timestamp). Entries are processed in descending (D) or ascending (T) order of date.
X or E	Fileid extension. i.e. The qualifier following the last dot/period. Entries are processed in descending (X) or ascending (E) alphabetical order.
M or N	Fileid name. i.e. All qualifiers preceing the last dot/period. Entries are processed in descending (M) or ascending (N) alphabetical order.
H or P	Fileid path. i.e. The directory path including the fileid name. Entries are processed in descending (H) or ascending (P) alphabetical order.
S or Z	File size. Entries are processed in descending (S) or ascending (Z) order of size.
0 or NO	Unsorted in the order returned by the system.

The default directory input order is 0 (unsorted) which may be overridden by a SORTDIR specification on a READ operation.

SSN *source*
Applicable to ODBC table input and output, SSN specifies the ODBC data source name (*source*) to which an ODBC connection will be made. Note that specification of an ODBC source name that has been defined to the local system, is mandatory for ODBC table input.

The *source* value must be specified as a character constant which may be overridden by an SSN parameter specified on a READ or WRITE operation.

SUBDIR *int* | **NOSUB**
SUB
Applicable to directory record input via the READ DIR or READ DIRDATA operation for Windows and Unix-like file directories only.

SUBDIR specifies an integer constant value (*int*) which represents the number of levels of nested sub-directory entries to be included for input. e.g. SUBDIR 1 will only read directory entries for files in the input directory, and in all sub-directories named in the input directory.

The default value is 0 (entries in the input directory only) which may be overridden by a SUBDIR specification on a READ operation. The SUBDIR option may be specified without *int*, in which case the default value is 255. NOSUB is a synonym for SUBDIR 0.

UNPRINTABLE *char_string*
Nominates character code points that are to be treated as unprintable by the LOG, PLOG and PRINT operations.

The SELCOPY print/log types that involve interpretation of code points as being printable or unprintable are: character (C), dump (D) and mixed character and hex (M, MC or MP). For these print/log types, only the code points identified as being printable will be passed without translation to the output stream.

The *char_string* is a character constant representing one or more character points to be treated as unprintable. e.g. To set as unprintable the ASCII characters "<" (less than) and "=" (equals) at code points x'60' and x'61' respectively, set `UNPRINTABLE X'6160'`.

Default unprintable code points are illustrated by the description of TYPE C output for the PRINT operation.

USER *userid*
Applicable to ODBC table input and output, USER specifies the user name (*userid*) used to connect to the data source via ODBC. *userid* must be specified as a character constant.

The *userid* value may be overridden by specification of the USER parameter on a READ or WRITE operation.

WORKLEN *int*
WORKAREA
WORKA
W

Valid in program control statements only, OPTION WORKLEN specifies an integer constant value *int* and indicates that a user work area of length *int* bytes is to be defined. The work area will be initialised so that all positions contain the character specified by option FILL in the program control statements.

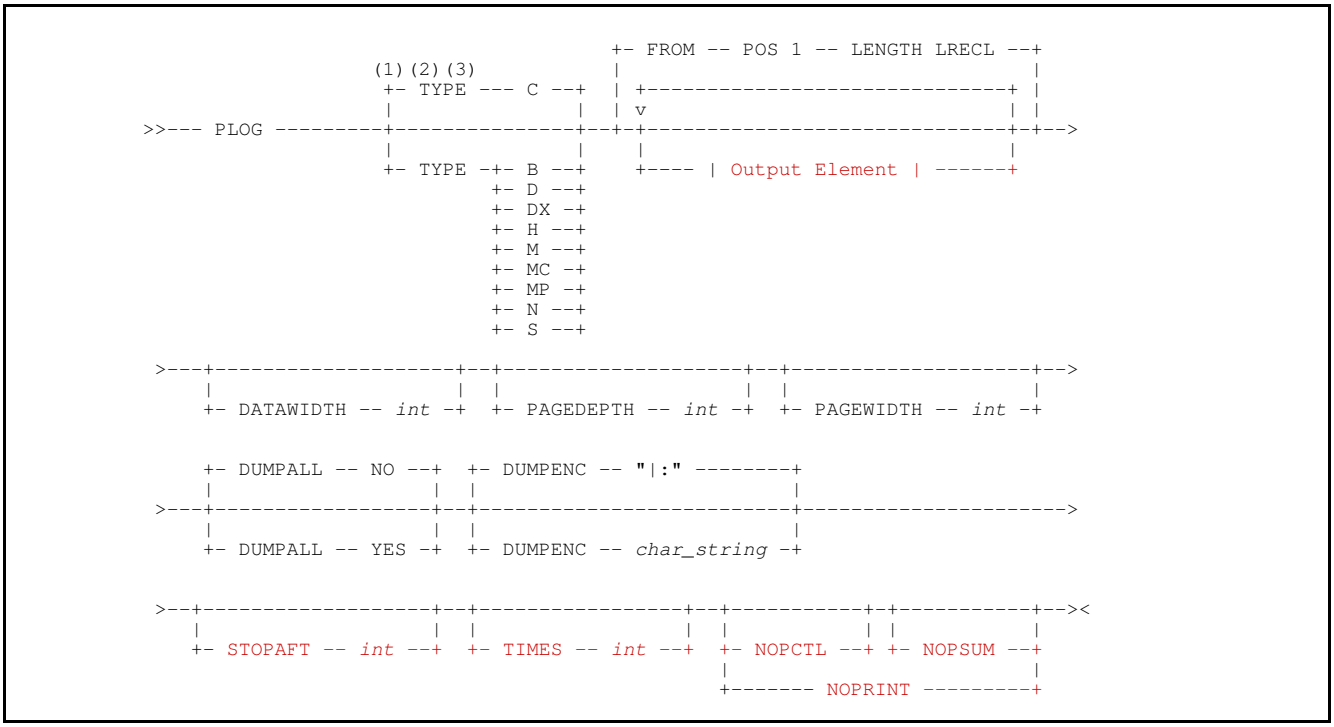
Alternatively, option WORKLEN may be specified on a READ operation for the prime input object. However, WORKLEN may not be set more than once in the same SELCOPY program.

If there is no input object, an 80 character work area is defined by default. Otherwise, no work area is defined and input data is referenced directly from the data object input buffer.

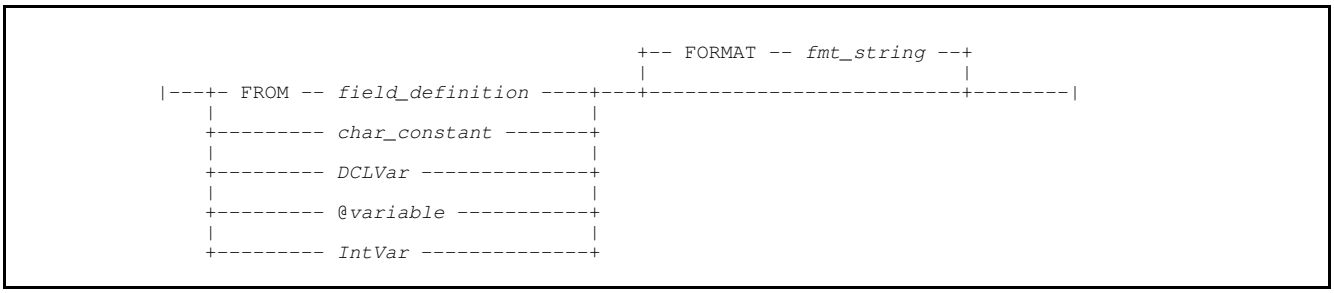
PLOG

Print data to the SELCOPY list output and write the same data to the log output.

Syntax:



Output Element:



Syntax Notes:

(1) If only one output element is used, then TYPE may be specified following the output element specification.

(2) The print output type defaults to TYPE C unless only one output element is specified, and that output element is a field definition with TYPE B specified. If this is the case, the default print type is also TYPE B.

(3) TYPE is ignored if the REPORT option is in effect for SELCOPY's report style print output.

Description:

Equivalent to executing both **PRINT** and **LOG** operations for the same output data line. The same output format, as specified by TYPE, is used for both the print and log output.

PLOG syntax and defaults are based on the PRINT operation. Differences between the PLOG and LOG operations are as follow:

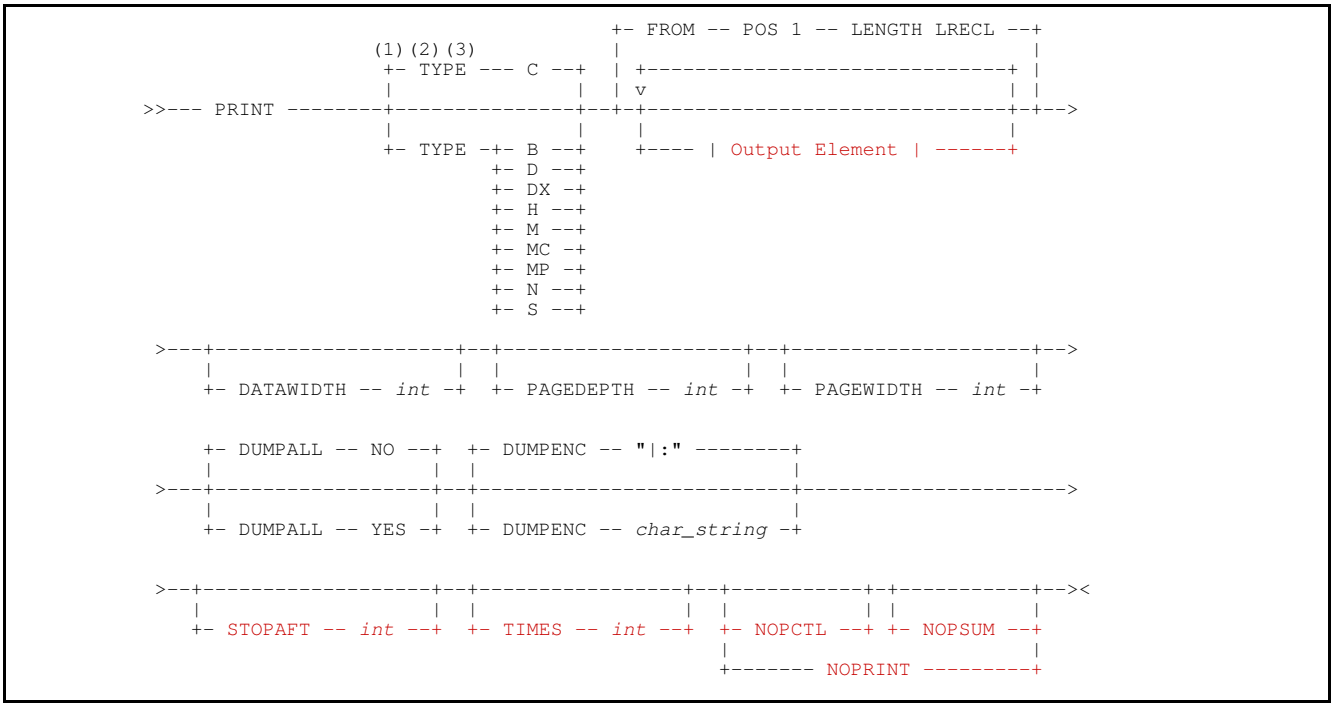
1. PLOG does not support the REPLY parameters for interactive input.
2. The default output data format is character (TYPE C), not system (TYPE S).

See the PRINT and LOG operations for information on parameters and output destination.

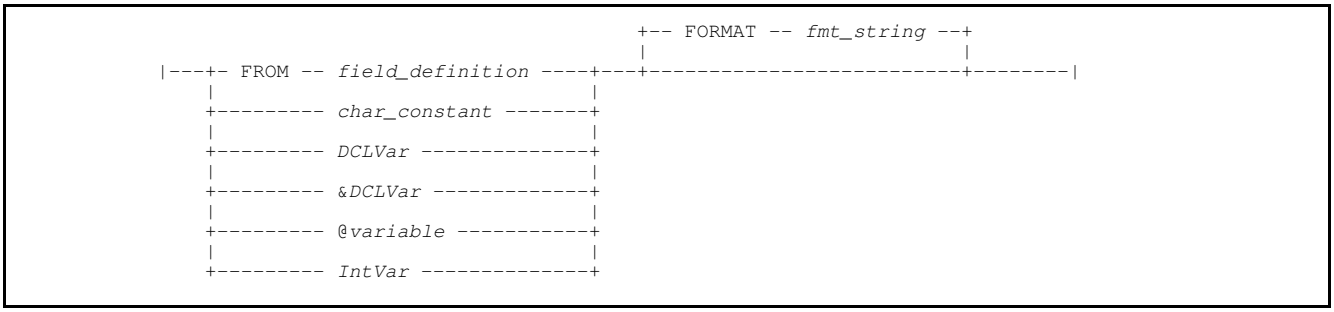
PRINT

Print data to the SELCOPY list output.

Syntax:



Output Element:



Syntax Notes:

(1) If only one output element is used, then TYPE may be specified following the output element specification.

(2) The print output type defaults to TYPE C unless only one output element is specified, and that output element is a field definition with TYPE B specified. If this is the case, the default print type is also TYPE B.

(3) TYPE is ignored if the REPORT option is in effect for SELCOPY's report style print output.

Synonyms:

PRINT	PR PRT SYSPRINT

Description:

Each execution of the PRINT operation will write one or more lines of data to the SELCOPY list output. The list output may be a file identified by input parameter `-lst` with a default of stdout for Windows and POSIX environments or SYSPRINT otherwise.

The length of the printed data is determined by the combined display lengths of the output elements. If environment option REPORT is not in effect, then, for all types of print output other than TYPE S, the printed data will be formatted within the Print

Block section of the SELCOPY list output.

Data in the print block is formatted into columns which, in addition to the printed data, contain information such as the prime input record number, the PRINT operation selection id and its execution number. Except for the printed data column which is preceded by a scale counting guide, each print block column is preceded by a standard, underlined column name header. These column headers are also written following a change to or from dump (TYPE D or DX) format print output and before continuing a print block on a new page of the list output.

Each row of values in the print block may occupy one or more lines of the list output. e.g. Print TYPE B format (both character and hex representation) occupies one line for the character representation, 2 lines for the up-down hex representation of the printed data and 1 line of blank characters.

The DATAWIDTH parameter determines the data width of the printed data column. If the data width is less than the length of printed data, then the printed data will wrap onto the next row of the printed data column. Each new row of wrapped data is preceded by a signed numeric value indicating the offset from the start of the printed data.

```
INPUT   SEL SEL                                                                                                  1  RECORD
RECNO   TOT ID.          1         2         3         4         5         6         7         8         9         0  LENGTH
-----   --- --- ....,....0...,....0...,....0...,....0...,....0...,....0...,....0...,....0...,....0...,....0...,....0  ------

    1     1   2 CB12DD714D51828C00A208CRolling In the Deep                                                          407
              CCFFCCFFFCFFFFFCFFCFFFCD9998984C94A884C8894444444444444444444444444444444444444444444444444444444444444
              3212447144518283001208396339570950385045570000000000000000000000000000000000000000000000000000000000000

     +100                                        Adele
              4444444444444444444444444444444444444444444C8898444444444444444444444444444444444444444444444444444444444
              0000000000000000000000000000000000000000000014535000000000000000000000000000000000000000000000000000000000

     +200                21                                                                        :  :Q      201A
              4444444444444FF444444444444444444444444444444444444444444444444444444444444444444444444444007F07D700410FFFC
              0000000000000210000000000000000000000000000000000000000000000000000000000000000000000003AD0A861040C2011

     +300           00AAdele                                    2011-01-21T08:00:00Z2012-08-02T11:30:36Z2011-08-19T12
              011FFCC8898444444444444444444444444444444444444FFFF6FF6FFEFF7FF7FFEFFFF6FF6FFEFF7FF7FFEFFFF6FF6FFEFF
              55C0011453500000000000000000000000000000000000020110010213 08A00A0092012008002311A30A3692011008019312

     +400       :29:17Z
              7FF7FFE
              A29A179

    2     2   2 2648A25633D1540400B208ERumour Has It                                                             407
              FFFFCFFFFFCFFFFFFFCFFFCDA99A94C8A4CA444444444444444444444444444444444444444444444444444444444444444444444
              2648125633415404002208594464908120930000000000000000000000000000000000000000000000000000000000000000000

     +100                                        Adele
              4444444444444444444444444444444444444444444C8898444444444444444444444444444444444444444444444444444444444
              0000000000000000000000000000000000000000000014535000000000000000000000000000000000000000000000000000000000

     +200                21                                                                        ` i      201A
              4444444444444FF444444444444444444444444444444444444444444444444444444444444444444444444444006207380410FFFC
              0000000000000210000000000000000000000000000000000000000000000000000000000000000000000003820919 1040C2011

     +300         <00AAdele                                    2011-01-21T08:00:00Z2012-08-02T11:30:36Z2011-08-19T12
              044FFCC8898444444444444444444444444444444444444FFFF6FF6FFEFF7FF7FFEFFFF6FF6FFEFF7FF7FFEFFFF6FF6FFEFF
              77C0011453500000000000000000000000000000000000020110010213 08A00A0092012008002311A30A3692011008019312

     +400       :30:11Z
              7FF7FFE
              A30A119
              ....,....1...,....2...,....3...,....4...,....5...,....6...,....7...,....8...,....9...,....0
```

Figure 14. PRINT TYPE B output with DATAWIDTH 100.

For dump (TYPE D or DX) print format output, the printed data occupies the entire row of data written to the print block, including areas used by the non-printed data columns. Therefore, for dump format output, the data width is not determined by DATAWIDTH, but the PAGEWIDTH parameter. e.g.

```
  INPUT   SEL SEL
  RECNO   TOT ID.
  -----   --- ---
      1     1   2           407
0000  C3C2F1F2 C4C4F7F1 F4C4F5F1 F8F2F8C3  F0F0C1F2 F0F8C3D9 96939389 958740C9     |CB12DD714D51828C00A208CRolling I|
0020  9540A388 8540C485 85974040 40404040  40404040 40404040 40404040 40404040     |n the Deep                       |
0040  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040     |                                 |
0060       =same=                                                                  :                                 :
0080  40404040 40404040 40404040 404040C1  84859385 40404040 40404040 40404040     |               Adele             |
00A0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040     |                                 |
00C0  40404040 40404040 40404040 40404040  40404040 40F2F140 40404040 40404040     |                     21          |
00E0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040     |                                 |
0100  40404040 40404040 40404040 40404040  40404040 40404040 40404000 037AFD00     |                              ..:..|
0120  7AD87601 0044100C F2F0F1C1 05151CF0  F0C1C184 85938540 40404040 40404040     |:Q......201A...00AAdele          |
0140  40404040 40404040 40404040 40404040  40404040 40404040 404040F2 F0F1F160     |                             2011-|
0160  F0F160F2 F1E3F0F8 7AF0F07A F0F0E9F2  F0F1F260 F0F860F0 F2E3F1F1 7AF3F07A     |01-21T08:00:00Z2012-08-02T11:30:|
0180  F3F6E9F2 F0F1F160 F0F860F1 F9E3F1F2  7AF2F97A F1F7E9                         |36Z2011-08-19T12:29:17Z          |
      2     2   2           407
0000  F2F6F4F8 C1F2F5F6 F3F3C4F1 F5F4F0F4  F0F0C2F2 F0F8C5D9 A49496A4 9940C881     |2648A25633D1540400B208ERumour Ha|
0020  A240C9A3 40404040 40404040 40404040  40404040 40404040 40404040 40404040     |s It                             |
0040  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040     |                                 |
0060       =same=                                                                  :                                 :
0080  40404040 40404040 40404040 404040C1  84859385 40404040 40404040 40404040     |               Adele             |
00A0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040     |                                 |
00C0  40404040 40404040 40404040 40404040  40404040 40F2F140 40404040 40404040     |                     21          |
00E0  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040     |                                 |
0100  40404040 40404040 40404040 40404040  40404040 40404040 40404000 03682200     |                             .....|
0120  79318901 0044100C F2F0F1C1 07474CF0  F0C1C184 85938540 40404040 40404040     |`.i.....201A..<00AAdele          |
0140  40404040 40404040 40404040 40404040  40404040 40404040 404040F2 F0F1F160     |                             2011-|
0160  F0F160F2 F1E3F0F8 7AF0F07A F0F0E9F2  F0F1F260 F0F860F0 F2E3F1F1 7AF3F07A     |01-21T08:00:00Z2012-08-02T11:30:|
0180  F3F6E9F2 F0F1F160 F0F860F1 F9E3F1F2  7AF3F07A F1F1E9                         |36Z2011-08-19T12:30:11Z          |
```

Figure 15. PRINT TYPE D output with PAGEWIDTH 140.

Parameters:

`Output Element`
Specifies an output element that constitutes a portion of the printed record data.

Multiple output element specifications are concatenated with no intervening blanks and in the order in which they are specified to construct the complete print record. The combined lengths of each of the elements define the output printed record length.

If no output element is specified, the output record data defaults to a single field definition starting at position 1 with length equal to the prevailing value of the variable LRECL. (i.e. FROM POS 1 LENGTH LRECL).

Before data is printed, output element values that are of numeric or character numeric data type are automatically converted to decimal character display format if any of the following are true:

1. The output element is a *DCLVar*.
2. Multiple output elements are specified.
3. FORMAT is included in the output element specification.

Note that &*DCLVar* may be used to output the unformatted value of a numeric *DCLVar*.

Each output element may be specified in one of the following formats:

`FROM` *`field_definition`*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type.

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL. However, if parameter FORMAT is used to convert the field data to printable hex, specification of a length is mandatory.

Applicable only to PRINT, LOG and PLOG operations, keyword LRECL, which usually defines the maximum output record length, is instead used as a synonym for keyword LENGTH in a type 1 field definition.

Although not necessary, keyword FROM may also be used before an output element specified as a *char_constant* or a *DCLVar* of character data type.

`char_constant`
A quoted or hexadecimal character constant.

`DCLVar`
The name of a previously defined declared variable of any data type.

`&DCLVar`
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

`@variable`
The name of an @variable that has a non-null value.
If @*variable* is null, return code 8 is set and the element value of "*?*" is used.

`IntVar`
The name of a SELCOPY internal variable.

`FORMAT` *`fmt_string`*
`FMAT`
`FMT`
FORMAT specifies *fmt_string*, a format string used as the character display template for data specified by the output element.

The source data in an output element of numeric data type is automatically converted to displayable character format (using a CVxC operation) before it gets printed. The length of the output element is determined by *fmt_string*, not the length of the output element source.

For output elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For output elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the output element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For an output element specified as an @*variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
`'SS,SSS,SSS,SS9'`	Used for @*variable*, *IntVar* and *DCLVar* representing an integer value.
`'SS,SSS,SSS,SS9.9999'`	Used for *DCLVar* representing a rational value.

DATAWIDTH *int*
DW

Applicable to all types of print format other than system (TYPE S) or dump (TYPE D or DX), DATAWIDTH specifies the width of the printed data column in the print block. i.e. the length of printed data to be written to each line of the SELCOPY list output.

DATAWIDTH is an environment option which is established during control statement analysis and applies to all PRINT and LOG operations executed during the SELCOPY program run. See OPTION DATAWIDTH for further details.

The default is the value set by OPTION DATAWIDTH in the SELCNAM file, otherwise a value of 100 is used.

DUMPALL YES | **NO**

Applicable to dump (TYPE D or DX) format only.

DUMPALL determines whether or not second and subsequent consecutive lines of dump format output that contain identical data to the first line, are to be condensed and displayed as a single output row containing the literal "=same=" followed by a parenthesised count of the number of condensed lines.

For the PRINT operation on which it is specified, DUMPALL YES will output all line containing duplicate data, DUMPALL NO will condense these lines. Note that DUMPALL is a synonym for DUMPALL YES.

The default value is set by environment option, DUMPALL. Otherwise, the default is NO.

DUMPENC *char_string*

Applicable only to dump format which includes character representation of the data (TYPE D).

For the PRINT operation on which it is specified, DUMPENC specifies *char_string*, a character constant of length 1 or 2, which identifies the characters used to enclose the character representation of the dump format output.

1. The first (or only) character defines the enclosing character used on print lines that have not been condensed to a single line. (See option DUMPALL above).

2. The second character defines the enclosing character used on print lines lines that have been condensed to a single line.

The default value is set by environment option, DUMPENC. Otherwise, "|" (or symbol) and ":" (colon) are used as the first and second characters respectively.

NOPCTL, NOPSUM, NOPRINT

See common parameters NOPCTL, NOPSUM, NOPRINT for details.

PAGEDEPTH *int*
PD

Specifies the depth (number of lines) of each page of the SELCOPY list output.

For all types of print format other than system (TYPE S), if output of a print row would exceed the PAGEDEPTH value (minus 1 for the print block scale line footer), a new page is started. At the start of each new page and before output of the print row data, the page header lines are written (so long as environment option HEAD is not set to NO), followed by the print block headers. Thus, the print block headers and footers are displayed on each page.

PAGEDEPTH is an environment option which is established during control statement analysis and applies to all SELCOPY list output. See OPTION PAGEDEPTH for further details.

The default value is set by OPTION PAGEDEPTH in the SELCNAM file, otherwise a value of 58 is used.

PAGEWIDTH *int*
PW

Specifies the width of each page of the SELCOPY list output which, for dump print format (TYPE D or DX) only, restricts the length of printed data written to each line of the print block.

See description of TYPE=D in the print block output for details on how PAGEWIDTH affects dump format print.

PAGEWIDTH is an environment option which is established during control statement analysis and applies to all PRINT and LOG operations executed during the SELCOPY program run. See OPTION PAGEWIDTH for further details.

The default value is set by OPTION PAGEWIDTH in the SELCNAM file, otherwise a value of 132 is used for POSIX environments and 133 for z/OS and z/VM CMS environments..

STOPAFT *int*

See common parameter STOPAFT for details.

TIMES *int*

See common parameter TIMES for details.

TYPE B | **C** | **D** | **DX** | **H** | **M** | **MC** | **MP** | **N** | **S**
TY

Identifies the type of print output to be used when writing the printed data.

TYPE B

Print both character (TYPE C) and hex (TYPE H) format output. However, unlike TYPE C print format, an unprintable character is displayed as a blank.

Each row of printed data in TYPE B format occupies 4 lines of the list output: 1 line for the character representation, 2 lines for the up-down hex representation and 1 line of blank characters. e.g.

```
     }   n  L  CBLATRACV3R4M0 ~201511121633
000071CD68C400444455445353430733333333333330000
0001D4D5E53C0032C142136324D00E2015111216330000
```

TYPE C

Print character format output.

Each row of printed data in TYPE C format occupies 1 line of the list output and only characters within the printed data that are determined as being printable are written without translation to the list output print block. Each character, determined as being unprintable, is translated to a "." (dot/period) before the write occurs. e.g.

```
....}...n..L..CBLATRACV3R4M0.~201511121633....
```

The code points treated by SELCOPY as being printable or unprintable, may be tailored using the PRINTABLE and UNPRINTABLE options.

The following table details the default interpretation ("Y" = printable, "." = unprintable) of each code point on **EBCDIC** based systems. (e.g. z/OS and z/VM CMS).

HEX	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
3-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
4-	Y	.	.	.	.	.	.	.	.	.	Y	Y	Y	Y	Y	Y
5-	Y	.	.	.	.	.	.	.	.	.	Y	Y	Y	Y	Y	Y
6-	Y	Y	.	.	.	.	.	.	.	.	Y	Y	Y	Y	Y	Y
7-	.	.	.	.	.	.	.	.	.	Y	Y	Y	Y	Y	Y	Y
8-	.	Y	Y	Y	Y	Y	Y	Y	Y	Y	.	.	.	.	.	.
9-	.	Y	Y	Y	Y	Y	Y	Y	Y	Y	.	.	.	.	.	Y
A-	.	Y	Y	Y	Y	Y	Y	Y	Y	Y	.	.	.	Y	.	.
B-	.	.	.	.	.	.	.	.	.	.	Y	.	Y	Y	.	.
C-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	.	.	.	.	.	.
D-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	.	.	.	.	.	.
E-	Y	.	Y	Y	Y	Y	Y	Y	Y	Y	.	.	.	.	.	.
F-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	.	.	.	.	.	.

Similarly, the following table details the default interpretation ("Y" = printable, "." = unprintable) of each code point on **ASCII** based systems.

HEX	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
2-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
3-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
4-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
5-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
6-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
7-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	.
8-	Y	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
9-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
A-	.	.	.	Y	.	.	Y	.	.	.	.	.	Y	.	.	.
B-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
C-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
D-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
E-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
F-	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

TYPE D

Print dump format output with both displayable hex and character representation.

Each row of printed data occupies 1 line of the list output and includes a 4 character hexadecimal numeric value identifying the offset of the data in the current row from the start of the printed data. This is followed by the displayable hex and then the character representation of the same length of printed data. The length of data represented by each print row is determined by the prevailing PAGEWIDTH value. e.g.

```
0000  00000001 7D14CDD5 6E85C34C 00004342     |....}...n..L..CB|
0010  4C415452 41435633 52344D30 007E3230     |LATRACV3R4M0.~20|
0020  31353131 31323136 33330000 0000         |1511121633....  |
```

Displayable hex is comprised of a number of groups of 8 hex characters, each group representing 4 bytes of data. Each byte of data is represented by an adjacent pair of hex characters located at even offsets from the start of the displayable hex data.

Character representation of the printed data follows the displayable hex and is enclosed in delimiter characters as defined by the environment option DUMPENC (default "|", the OR symbol). Printable and unprintable characters are as described for the TYPE C print format and, like TYPE C, each unprintable character is displayed as a "." (dot/period).

If environment option DUMPALL=NO is in effect, consecutive rows of dump format print output that contain the same data will be grouped and condensed. To do this, SELCOPY outputs the first row of the group in the standard format printed followed by a single row which represents the remaining rows of the group. This row contains the next hex value offset and, in place of the displayable hex, the literal "=same=" followed by a parenthesised count of the number of print lines represented by the row. No character representation of the data is displayed, however, the character data delimiter characters are included using the second DUMPENC environment option character (default ":", colon).

See TYPE=D Output in the description of the print block for more detailed information.

TYPE DX

Print dump format output (TYPE D) but containing displayable hex only. e.g.

```
0000  00000001 7D14CDD5 6E85C34C 00004342
0010  4C415452 41435633 52344D30 007E3230
0020  31353131 31323136 33330000 0000
```

TYPE H

Print hex format output.

Each byte is represented by 2 digit hexadecimal value in the range x'00' to x'FF'. Hex print format displays each byte of the printed data as its hexadecimal value.

Each row of printed data in TYPE H format occupies 3 lines of the list output. The first line displays the first digit of each byte of printed data and, aligned directly below it, the second line displays the second digit so that the hex value is displayed vertically in an up-down format. The third line contains only blank characters. e.g.

```
000071CD68C400444455445353430733333333333330000
0001D4D5E53C0032C142136324D00E2015111216330000
```

TYPE M or TYPE MC or TYPE MP

Print mixed character (TYPE C) and hex (TYPE H) format output.

For mixed print format, each individual byte of the printed data will be displayed in either TYPE C or TYPE H format. To display all printed data in both formats simultaneously, TYPE B should be used.

SELCOPY examines each byte of the printed data and, if it is printable (as indicated by the description for TYPE C), then character representation is used. Otherwise the byte is displayed as hex.

Each row of printed data in TYPE M format occupies 3 lines of the list output. For each byte of the printed data, the first line displays either the printable character representation or the first digit of the hexadecimal value. The second displays either a blank (if the character is printable) or the second digit of the hexadecimal value. The third line contains only blank characters. e.g.

```
0000}1CDn8CL00CBLATRACV3R4M00~2015111216330000
0001 4D5 53 00             0             0000
```

TYPE N

Print with no conversion of unprintable characters.

TYPE N format output is identical to TYPE C character print format except that **all** characters are considered printable. e.g.

```
} ÍÕn ÃLCBLATRACV3R4M0~201511121633
```

If the print output is spooled to a real printer device, then care should be taken that the unprintable characters do not perform unintentional printer control. e.g. 'x'0C' is interpreted as a printer control form feed instruction by a Windows raw printer driver.

TYPE S

System print with no conversion of unprintable characters. TYPE S format output is identical to TYPE N with the following exceptions:

1. All page headers are suppressed.
2. All print block headers and footers are suppressed.
3. All print block column values, other than the printed data, are suppressed.
4. Each row of printed data starts in the first column of the list output.
5. Rows of printed data do not wrap onto the next line of the list output.
6. The maximum length of each row of printed data is 133.

On z/OS and z/VM CMS systems, one of the following ASA characters is expected in the first column of the print data:

ASA Character	Description
1 (one)	Skip to top of page and print.
+ (plus)	Space 0 lines and print. (Used for overprinting)
b (blank)	Space 1 line and print.
0 (zero)	Space 2 lines and print.
- (minus)	Space 3 lines and print.

Invalid ASA characters are tolerated, however, valid ASA characters are used by SELCOPY to maintain the internal variable LINE. e.g. An ASA character of "1" will reset the value of LINE back to 1. An invalid ASA character is treated as being blank (i.e. space 1 line), and so the value of LINE is incremented by 1.

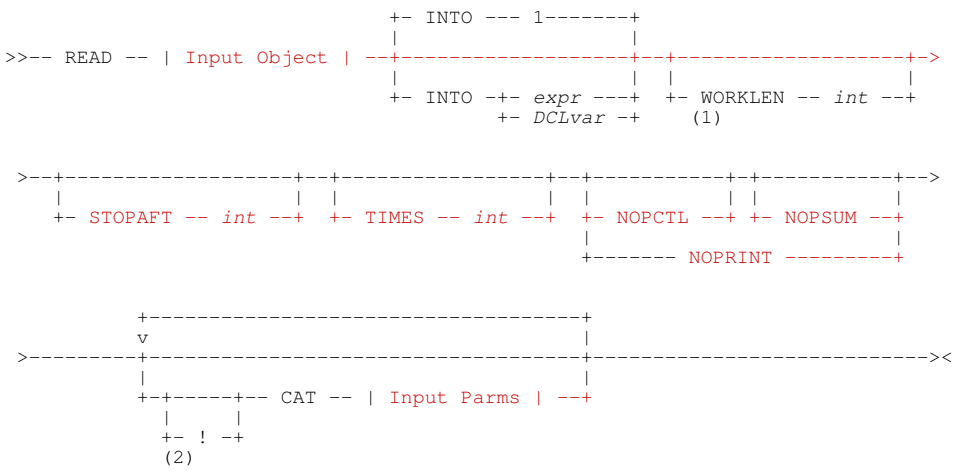
The print data record is treated as being of fixed length 133 record format (RECFM F, LRECL 133). Therefore, any output element specified as a field definition, will have a default length of 133 if one is not supplied (or implied) as part of the field definition. If a field assigned this default length references data extending beyond the length of the work area or current input record, then a length equal to the current value of variable LRECL is used instead.

Print data is truncated at 133 bytes and, if its total length (i.e. the sum of all the output element lengths) is less than 133, then it will be padded with blanks to length 133. Thereafter, if the SELCOPY list output file object is actually of variable length record format, any trailing blanks will automatically be truncated.

READ

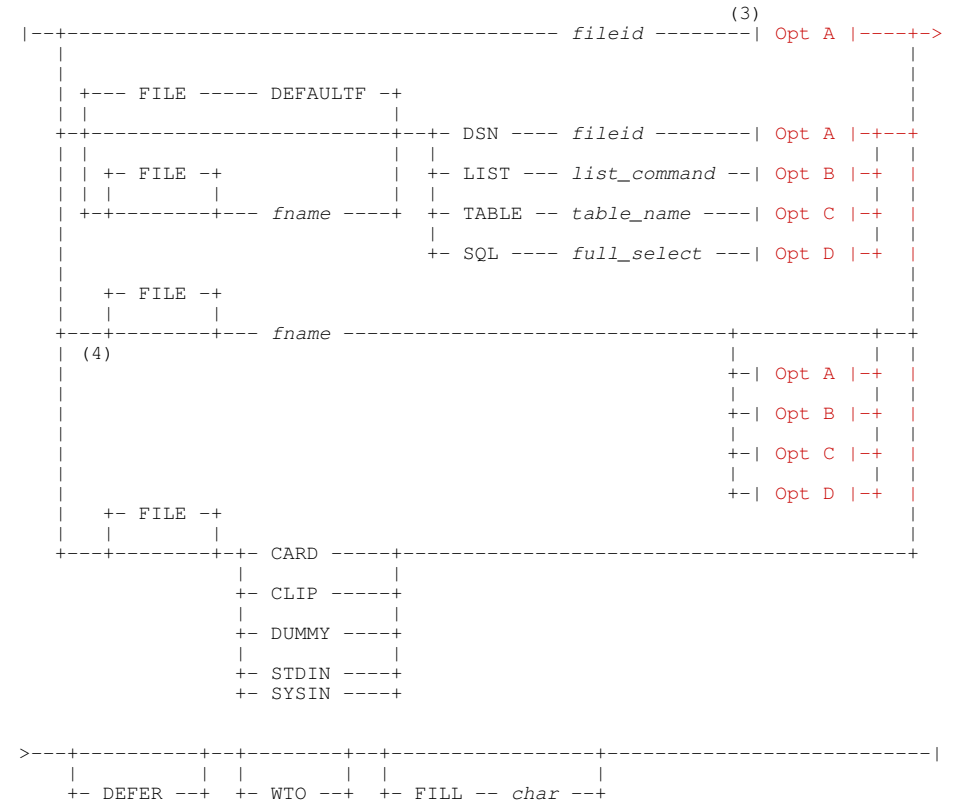
Inputs data from a data object source for processing by the SELCOPY program.

Syntax:



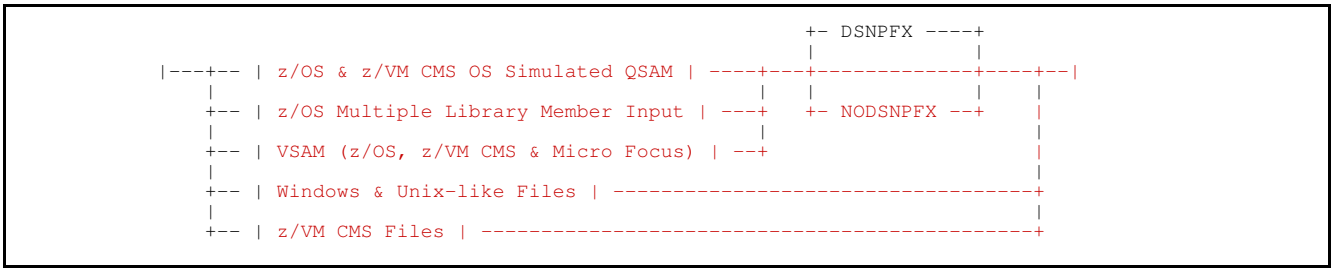
(1) WORKLEN is specified once only on either a READ or OPTION operation.
(2) Separator character (default !) required only if CAT is specified on the same control statement as READ.

Input Object: (Dataset/File, SELCOPYi List or Database Table)

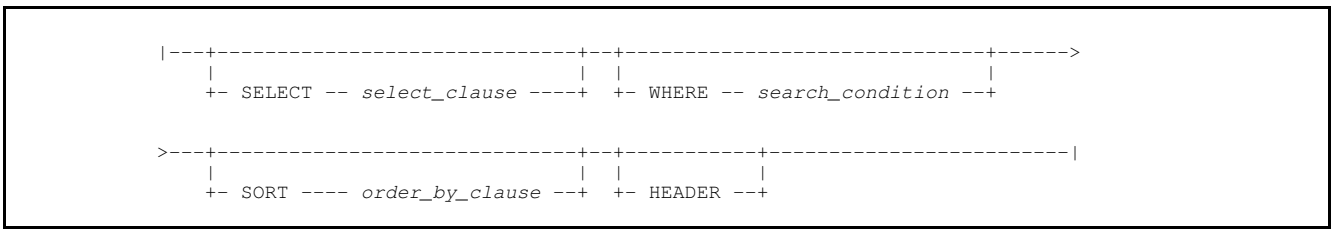


(3) Optional parameters may be specified on any READ, OPEN and CLOSE operations for the same *fname*.
(4) *fname* without an input object reference is valid if *fname* is already associated with a data object.

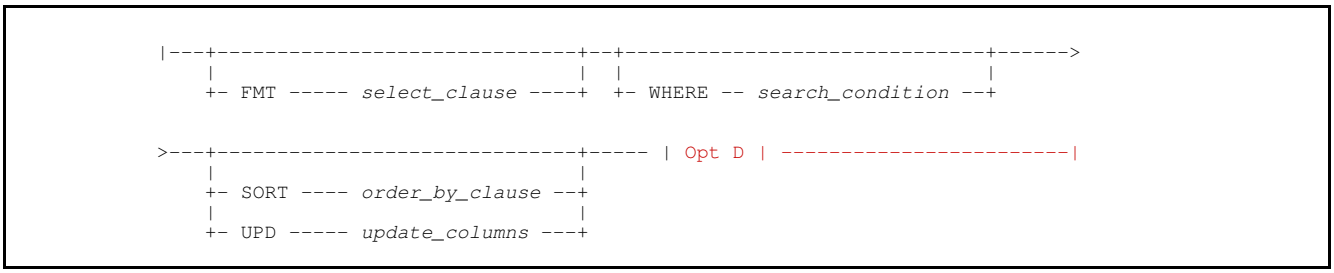
Opt A: (Dataset/File Object Options)



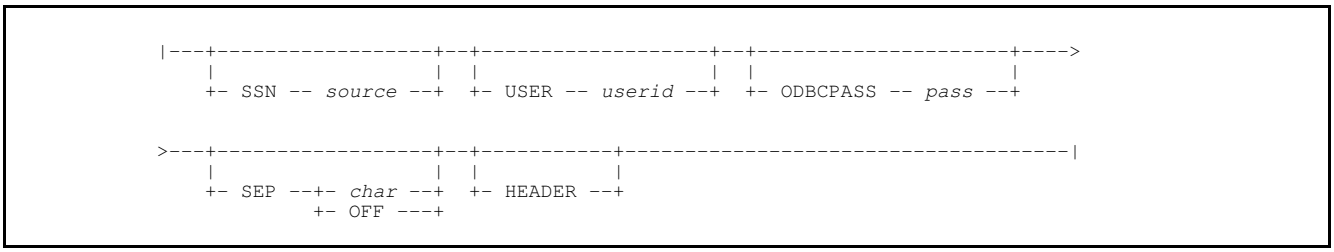
Opt B: (SELCOPYi List Object Options)



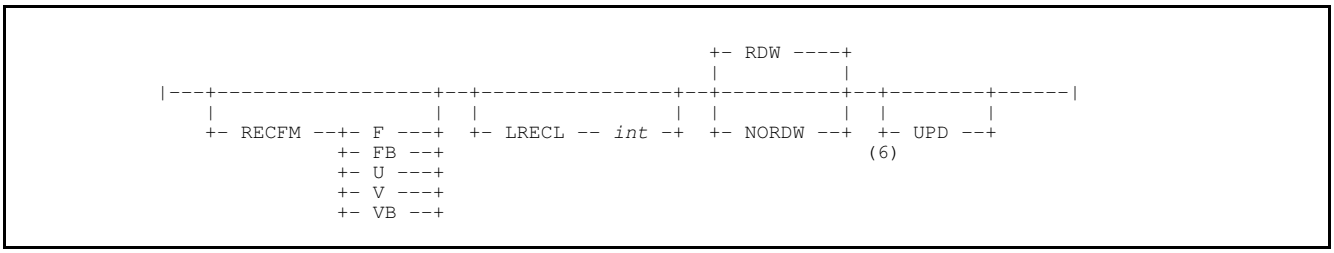
Opt C: (Database Table Object Options)



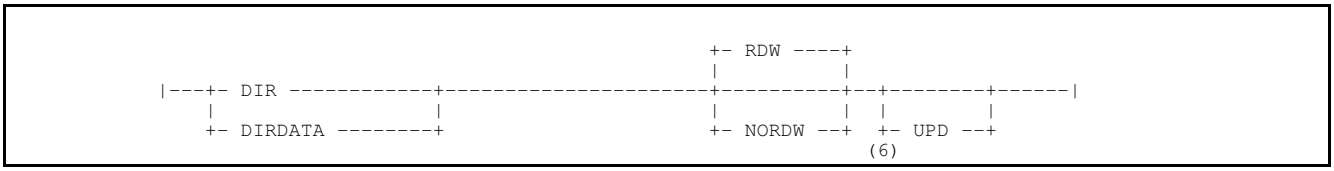
Opt D: (Common Database Table and SQL Query Object Options)



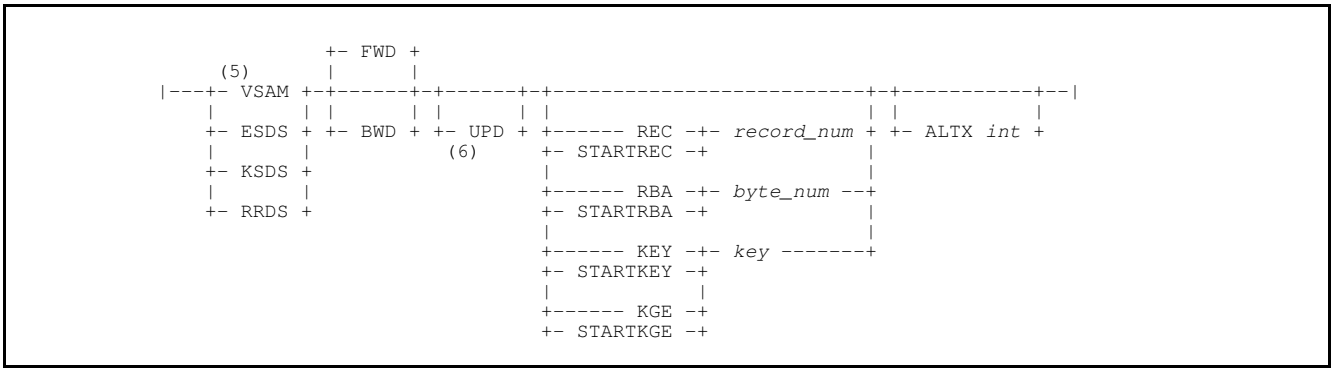
z/OS & z/VM CMS OS Simulated QSAM:



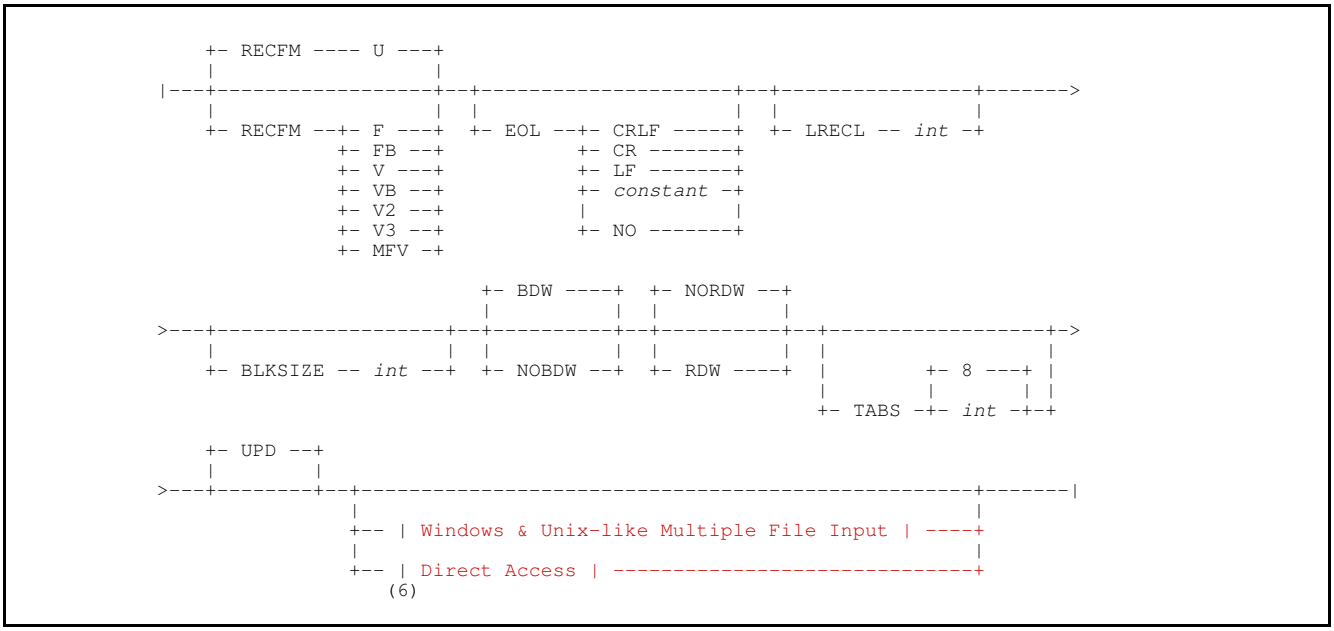
z/OS Multiple Library Member Input:



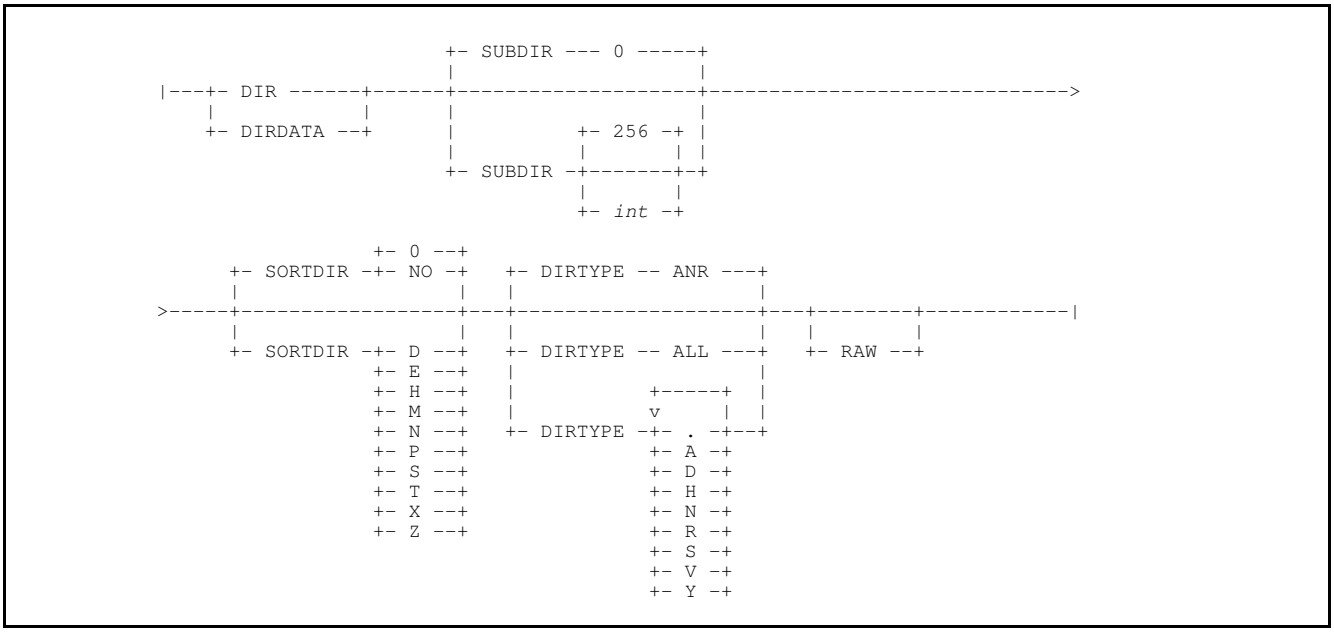
VSAM (z/OS, z/VM CMS & Micro Focus):



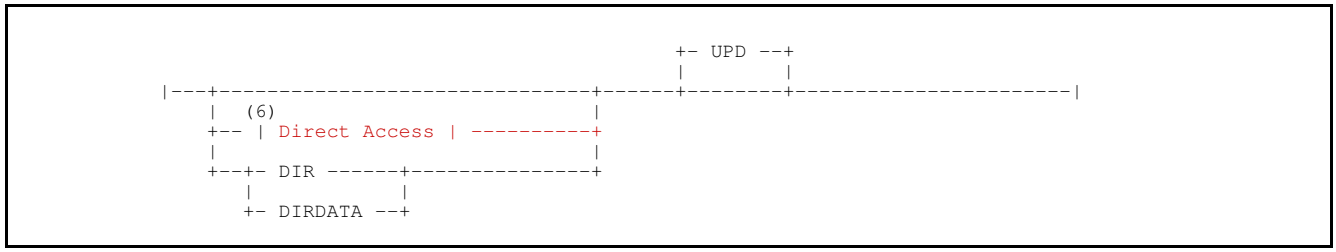
Windows & Unix-like Files:



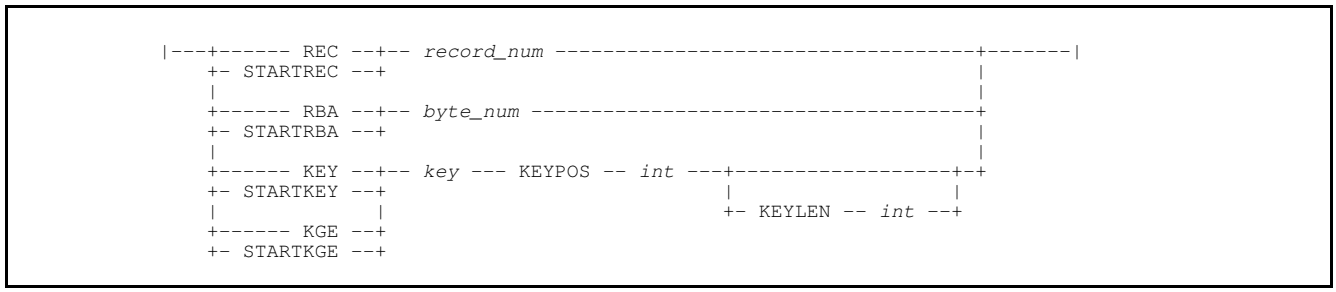
Windows & Unix-like Multiple File Input:



z/VM CMS Files:



Direct Access:



Syntax Notes:

(1) WORKLEN may be specified only once on either an OPTION operation or on the first occurrence of a READ operation within the SELCOPY input control statements.

(2) **!** Keyword CAT is a sub-operation and so must be the first identifier specified on a SELCOPY statement. Specification of the defined statement separator character (default "!") before a sub-operation keyword is required only if it is not the first identifier. i.e. the READ operation keyword and/or CAT sub-operation keywords exist on the same input control file record.

(3) Optional parameters may be specified on any READ, OPEN or CLOSE operation for a data object referenced by the same *fname*. If different option values relating to file geometry are specified for the same *fname* (e.g. RECFM, LRECL, BLKSIZE, VSAM), then the last value specified will be used.

(4) READ of *fname*, without a reference to an input data object, is valid only if *fname* has already been associated with a data object. i.e.

1. *fname* is an already allocated z/OS or z/VM CMS ddname.
2. Control statement analysis pocessing has already occurred for an OPEN, CLOSE or READ operation that associates *fname* with a data object.

(5) Specification of VSAM or specific VSAM file type is not necessary for z/OS VSAM data sets. For z/OS, the VSAM data set type is established dynamically from the ICF catalog.

(6) UPD and Direct Access parameters are not supported on CAT.

Synonyms:

READ	RD GET INPUT IN

Description:

Each execution of a READ operation will present the program with a single line of data read from the specified input data object. The length of this data is determined by the type of input object and input parameter options specified on READ.

For z/OS and z/VM CMS file input, each line of data is a record as defined by the data set or file organisation. For Windows and Unix-like files (including CARD, CLIP and STDIN input), a record is usually a line of data terminated by end-of-line characters, although fixed and other variable length formats are also supported. The row of data returned by SELCOPYi list and ODBC table input is of fixed length, determined by the column widths selected.

Sub-operation, CAT, may be specified immediately following a READ operation (or another CAT sub-operation) in order to concatenate input from another data object to that of the object specified by the previous READ or CAT. Once end-of-file has been reached for one input object, the next execution of READ will read data from the next, concatenated data object.

By default, records are read sequentially from the start of the data object. Where applicable, a direct read may be performed for a record located anywhere within a file object. Similarly, the direction of sequential processing of VSAM and Micro Focus file objects may be reversed using the BWD and FWD parameters.

Input processing will read a block of data from the specified input data object into a storage buffer defined for that *fname*. If no user work area is defined and input records are not to be assigned to a character declared variable (via INTO *DCLvar*), then each execution of READ will update the base address (position 1) so that it points to the next, unprocessed record within the buffer.

If INTO *DCLvar* is specified and *DCLvar* is a character declared variable, then the input record data is a value which gets assigned to *DCLvar*. Otherwise, the line of data is copied to a position in the work area buffer (default position 1), where the position 1 base address is static.

Following any successful READ operation, the internal variable, LRECL, is automatically assigned a value equal to the length of the record read.

Parameters specified on READ and CAT apply either to the object open, data retrieval from the object, the execution of the individual SELCOPY operation or the definition of a user work area.

Object Open Parameters

By default, an input data object on READ or CAT is opened when the statement containing the operation keyword is processed during control statement analysis. Parameters affecting object open are applied once when the object is opened.

Open parameters, specified on READ or CAT operations for the same *fname* that occur in subsequent control statements, may update or expand on the values set by parameters in the first input operation. However, if the object is already open, they will have no effect unless the object is re-opened. Open parameters are as follow:

fileid	ESDS	RRDS	TABLE
DEFER	FMT	SELECT	UPD
DSN	KSDS	SORT	VSAM
DSNPFX	LIST	SQL	WHERE

Open of a data object assigned to *fname* may be deferred until selection time processing. Furthermore, the object may be closed and re-opened if required. In these cases, SELCOPY uses the prevailing values of open parameters specified on all input operations for *fname*.

Deferred open may be achieved using any of the following methods:

- Specify DEFER on the first READ operation for *fname*.
- Include an OPEN operation for *fname* on a control statement that occurs before the first READ operation for *fname*.
- Specify a dynamic value (i.e. a declared variable name or field definition) for the input data object specification parameters: *fileid*, DSN, FMT, LIST, SELECT, SORT, SQL, TABLE, UPD (for ODBC table input) and WHERE.

Data Retrieval Parameters

Data retrieval is performed when the READ operation is executed during selection time processing and, for Windows and Unix-like files, may involve additional processing by SELCOPY to establish the record and its length.

Data retrieval parameter values are established during control statement analysis. Therefore, parameter values specified on READ or CAT operations for the same *fname* that occur on subsequent control statements, may update or expand on values set by previous input operations. The prevailing parameter values are used for all data retrieval performed for *fname*. Data retrieval parameters are as follow:

BDW / NOBDW	EOL	RAW	SSN
BLKSIZE	FILL	RDW / NORDW	SUBDIR
DIR	HEADER	RECFM	TABS
DIRDATA	LRECL	SEP	USER
DIRTYPE	ODBCPASS	SORTDIR	

Operation Specific Parameters

Operation specific parameters apply only to the individual READ operation. With the exception of current input file position for sequential input, operation specific parameter values, specified on other READ or CAT operations for the same *fname*, do not affect the execution of the current READ operation. Operation specific parameters are as follow:

BWD	KEYLEN	REC	STARTREC
FWD	KEYPOS	STARTKEY	STOPAFT
INTO	KGE	STARTKGE	TIMES
KEY	RBA	STARTRBA	WTO

Parameters:

fileid

Identifies the READ operation as being for file input. *fileid* is a fileid clause specifying the name by which the input file is known to the local system.

If not specified as the DSN parameter value, then specification of FILE is invalid and the associated *fname* used by SELCOPY to reference input from *fileid* is derived from *fileid* itself. The derived fname is determined as follows:

- For z/OS, if *fileid* is the name of a sequential data set or a PDS/PDSE library and member, *fname* is set to be the first qualifier of the data set name.
- For z/VM, if *fileid* is the name of a CMS file (*fn ft fm*) , *fname* is set to be the file name qualifier (*fn*).
- For Windows and Unix-like files, the derived *fname* depends on the length of the portion of *fileid* that follows the last "." (dot/period), if any. In Windows, this is the fileid extension.
    1. If this length is 3 characters or less, *fname* is the up-to-8 character file name string that follows the path specification (if any) up to, but not including, "." (dot/period) prefixed fileid extension.
    2. Otherwise, *fname* is the character string which occupies up-to-8 characters of *fileid*, not including the path specification (if any).

On z/OS systems, if *fileid* is the name of a sequential data set or PDS/PDSE library and member, then the derived *fname* is also the ddname that SELCOPY dynamically allocates to *fileid*. (see FILE parameter).

fileid may contain wildcard characters to represent a fileid mask used for directory record (DIR/DIRDATA) input. A description of fileid mask specification is documented by the fileid clause under Input/Output Data Objects.

fileid may be specified as an unquoted literal, a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Notes on *fileid* specification:

1. If *fileid* is specified as a declared variable or field definition, then it must be an argument to parameter DSN. Also, open of the input file is deferred until the READ operation is executed during selection time processing.
2. In z/OS, where *fileid* may be interpreted as being either a z/OS data set name or an HFS/ZFS Unix-like file name, then SELCOPY will treat *fileid* first as a data set name and then, if not found, as a Unix-like file name.

   Similarly in a z/VM CMS environment, where fileid may be interpreted as either a CMS mini disk file specification or a BFS Unix-like file name.
3. On a z/OS system, if environment option DSNPFX is in effect (the default), then the TSO prefix (TSO/E) or ACF userid (batch) will be prefixed to *fileid* if the constant, variable or field definition value used is unquoted. Furthermore, if neither DSNPFX nor NODSNPFX is specified on the READ operation, open of a data set without a DSN prefix will be attempted if the initial attempt to open a data set with a DSN prefix fails.

   Only if this second attempt to open a data set also fails, will *fileid* be treated as a Unix-like HFS/ZFS file name as described in note 2. above.
4. A Windows or Unix-like *fileid* is not upper cased, even if specified as an unquoted literal.

ALTX *int*
KEYIX

Applicable only to Micro Focus VSAM input, ALTX identifies the alternate index number *int* to be used to process the indexed (KSDS) file. int must be specified as a decimal integer constant.

Default is ALTX 0. i.e. the primary index.

BDW | **NOBDW**

Specifies whether or not a Windows or Unix-like input file of variable length record format (RECFM V or VB) is blocked, with each block of records prefixed by a 4-byte block descriptor word (BDW).

Default is BDW, input of variable length record format is blocked.

BLKSIZE *int*
BLK
B

Specifies the size of the input buffer defined for *fname*.

BLKSIZE should never be specified for input of a z/OS data set, CMS file or IBM VSAM data set. For these types of file input, the input file buffer size is always determined by standard access method processing. However, if BLKSIZE is specified it restricts the maximum length of records read from the file which may result in a file geometry conflict error.

BLKSIZE is tolerated and ignored if specified on the READ operation for LIST and ODBC database table input.

For Windows and Unix-like files, including Windows clipboard (CLIP) and Micro Focus VSAM input, BLKSIZE overrides the default buffer size and, if LRECL is not specified, also imposes a maximum record length for the input record data. (see LRECL default). The input file buffer size is determined as follows:

1. The value specified on parameter BLKSIZE.
2. For fixed length record format (RECFM F) only, buffer size is the value specified on parameter LRECL, if greater than 2048.
3. 2048 bytes.

For Micro Focus VSAM input, if the buffer size is less than the maximum record length defined for the file, then ERROR 571 (Max lrecl exceeds buffer size) is returned.

For all input record formats, if the maximum record length imposed by the BLKSIZE value is less than the value specified for LRECL, then ERROR 571 (Bufsize < LRECL+delims) is returned.

For Windows and Unix-like files read with RECFM V or VB and option BDW, ERROR 537 (RECFM=V BDW/RDW value exceeds BLKSIZE or LRECL) is returned if the value in a BDW exceeds the explicit or implied BLKSIZE value.

BWD

Applicable only to z/OS and z/VM CMS KSDS and ESDS input, and also to Micro Focus VSAM input. BWD specifies that, when a sequential read is performed by the READ operation, the record that occurs before the current file input position will be read. i.e. in a backwards direction.

If *fname* has not been read since its open, then the first record returned by a backwards sequential read will be the last record in the file. The EOF condition is set when the file is read backwards and the current input position points to the first record.

For Micro Focus VSAM input and KSDS files only, multiple READ operations may be specified with BWD or FWD parameters to reverse the direction of sequential input for file *fname*. The direction of input may be reversed any number of times during the course of the program execution and so care should be taken not to cause a loop situation.

If BWD is specified on any READ operation, then either FWD or BWD must be specified on READ operations for the same *fname*.

The default input direction is forwards (FWD).

DEFER

Defer open of the data object until data is to be read during selection time processing.

By default an input file is opened when the statement on which the READ operation exists, is processed during control statement analysis.

DIR

Used to input the directory entries for z/OS library members or z/VM CMS, Windows or Unix-like files whose fileids match the fileid mask assigned to fname. Each directory entry is processed as a single input record.

For Windows and Unix-like files, directory records may also be filtered by DIRTYPE and SUBDIR parameter options and sorted using a SORTDIR parameter option. Furthermore, RAW may be specified to return the directory entry with no formatting applied.

UPD (update-in-place) processing is not supported on DIR input.

See also parameters DIRTYPE, SORTDIR and SUBDIR.

DIRDATA
DIRD
DD

Input the directory record, as described for DIR, followed by all data records for each file that matches the specified fileid mask.

The type of record last read for *fname* may be determined by an IF DIR/DATA condition.

Separate input record (INCOUNT) values are maintained for the directory and data records. The current value of INCOUNT for *fname* will apply to the type of record (directory or data) last read for *fname*. Therefore, before testing an INCOUNT value, it should first be established whether the last record read was a directory or data record.

The FLAG operation may be used to force early end of file, directory or disk for DIRDATA input so that the next record read will be a directory record for a new file in the same directory, in a new directory/sub-directory or in a new disk volume.

UPD (update-in-place) is supported on DIRDATA input for data records only. An attempt to update a directory record will return ERROR 565.

See also parameters DIRTYPE, SORTDIR and SUBDIR.

DIRTYPE *attributes*
TYPEDIR
TYPE

For DIR and DIRDATA input of Windows and Unix-like files only, DIRTYPE may be specified to filter directory entries based on the attributes of the files to which they refer. Only those file directory entries that match the requested file attributes are selected. Options are as follow:

Code	File Attribute required
.	The working "." and parent ".." directory entries.
R	Read-only.
H	Hidden.
S	System.
V	Volume-Id.
D	Subdirectory.
A	Archive. (Amended file.)
Y	Symbolic Link. (UNIX only.)
N	Normal. (No attribute flags set.)

Option keyword **ALL** may be specified to indicate that all directory entries are to be included.

Unless ALL is specified, multiple DIRTYPE options may be specified as an unquoted literal or as a quoted character constant. Blanks and comma characters may be specified and are ignored if the DIRTYPE options are supplied as a quoted character constant.

The default directory entry selection may be set using the DIRTYPE environment option, otherwise the default is 'A,N,R'.

DSN *fileid*

Specifies the name of the input file. See *fileid* for details.

If no *fname* is specified on a READ operation which uses DSN, then an *fname* of DEFAULTF is used by default.

DSNPFX | **NODSNPFX**
USERID | **NOUSERID**
UID | **NOUID**
DSNPFX YES | **DSNPFX NO**

Applicable only in z/OS where *fileid* identifies an unquoted data set name belonging to a PDS/PDSE library or a sequential or VSAM data set.

If *fileid* is an unquoted literal or an unquoted value identified by a declared variable or field definition, then DSNPFX will prefix the value with the one of the following:

◊ The ACF userid assigned to the job if running in batch.
◊ The current TSO prefix if running in a TSO/E environment.

If specified on the READ operation, DSNPFX, NODSNPFX or one of their synonyms will override the current value of the DSNPFX environment option for that input file only. If not specified on READ and DSNPFX (YES) is the environment default, SELCOPY will first attempt to open the file using a data set name prefix. If that fails, SELCOPY will attempt to open the file without a data set name prefix.

EOL **CRLF** | **CR** | **LF** | **NO** | *constant*

Applicable only to RECFM U input of Windows and Unix-like files, EOL defines the end-of-line characters for which SELCOPY will scan when de-blocking records from data in the input buffer.

The representation of each EOL argument is as follows:

EOL	ASCII	EBCDIC	Description
CRLF	X'0D0A'	X'0D15'	Default for Windows, z/OS HFS/ZFS and z/VM BFS.
LF	X'0A'	X'15'	Default for Linux and Unix file systems.
CR	X'0D'	X'0D'	
constant			A quoted or hex character constant of any length.
NO			No end-of-line characters are used.

By default, SELCOPY will scan for CRLF, LF and CR end-of-line characters as record delimiters.

EOL *constant* specifies a non-standard combination of characters to be used to delimit each input record. EOL NO indicates that records have no end-of-line characters, in which case the input record will be of length equal to the value currently assigned to the variable LRECL.

ESDS

Applicable to IBM VSAM data sets (z/OS and z/VM CMS) and Micro Focus VSAM files. ESDS identifies the input file as a VSAM ESDS (entry sequenced data set) or a Micro Focus variable length record sequential file.

For existing z/OS VSAM data sets, parameter ESDS does not need to be specified as SELCOPY automatically identifies the access method and file organisation of the input file.

For all Micro Focus indexed, relative and record sequential files, parameters VSAM, KSDS, ESDS and RRDS are synonymous and indicate only that input is to be performed via the Micro Focus file handler.

FILE *fname* | **CARD** | **CLIP** | **DUMMY** | **STDIN** | **SYSIN**
F

Identifies the name to be assigned to the input data stream. This name may then be used to reference the individual input data object within the SELCOPY control statements. See description of File Name for further details.

The value *fname* is a programmer defined name to be used for the file, SELCOPYi list or ODBC database table input specified by the DSN, LIST, TABLE or SQL parameter. It may only be specified as an unquoted literal.

On z/OS and z/VM CMS systems, *fname* may be a ddname which has already been allocated to a file object via a z/OS TSO ALLOC or JCL DD statement, or a z/VM CMS FILEDEF or DLBL command. If this is the case, no input data object reference (DSN, LIST, TABLE or SQL) is required on the READ operation.

If *fname* is not already an allocated ddname and DSN *fileid* references a z/OS sequential data set or PDS/PDSE library member, then SELCOPY will dynamically allocate *fname* to *fileid*. This is necessary since z/OS access methods require an allocated ddname to perform an open and subsequent I/O on a data set. By default the data set is allocated to *fname* with DISP=SHR.

Special instances of *fname* exist to identify the specific input data objects as follow:

CARD	Input is from the same source as the control statements. All records that follow the END operation are input to READ CARD. Unless SELCOPY program parameter -ctl has been specified, CARD, STDIN and SYSIN are synonymous.
CLIP	For Microsoft Windows only, input is from the Windows clipboard.
DUMMY	Input is from a dummy object source. No open is performed and the EOF condition is set following the first execution of READ DUMMY.
STDIN or **SYSIN**	Input is from the standard input stream (stdin). It is primarily used in Windows and POSIX environments to read output data piped from another application. STDIN and SYSIN are synonymous.

The *fname* value (or any of the special instances) may be specified with or without the FILE parameter keyword.

FILL *char*
PAD

If a work area has been defined, FILL specifies a single character, represented by a character constant, which will be used to pad an input record to the length of the previous record read from *fname*.

If the input record has a length which is greater than or equal to that of the previous record read, then no padding is required. Beware that, if a different INTO position is specified on READ operations for the same *fname*, record padding will occur for data at the input position to which the record is read. For example, in the following, padding will occur for data read into position 501 and data at position 1 is unchanged.

```
READ INDD dsn='c:\tmp\vardata.txt' INTO   1 FILL=X   * 1st record is long.
READ INDD                          INTO 501          * 2nd record is short.
```

The default pad character is set by environment option, FILL. Otherwise, the default is a blank.

FMT *select_clause*

Applicable to ODBC table input where a table or view is referenced via the TABLE parameter.

FMT specifies *select_clause*, a valid SQL select clause which includes the names of table columns (or expressions) that constitute the result table, any alternate column names to be used and whether duplicate rows are included (ALL or DISTINCT). The *select_clause* syntax is used as is to construct an SQL query statement for the input result table.

For details on *select_clause* syntax, please refer to SQL reference documentation for the DBMS to which the ODBC connection will be made.

The *select_clause* value may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *select_clause* is specified as a declared variable or field definition, then the object open, connection to the database and execution of the SQL query is deferred until the READ operation is executed during selection time processing.

FWD

Applicable only to z/OS and z/VM CMS KSDS and ESDS input, and also to Micro Focus VSAM input. FWD specifies that, when a sequential read is performed by the READ operation, the record that occurs following the current file input position will be read. i.e. in a forwards direction.

For Micro Focus VSAM input and KSDS files only, the direction of sequential input may be reversed for file *fname*. (See the BWD parameter)

Unless BWD has been specified on another READ operation for the same *fname*, specification of FWD is not necessary. The default input direction is forwards (FWD).

HEADER
HEAD
HDR
HD

Applicable to ODBC table and SELCOPYi list input, HEADER indicates that the two SELCOPY generated header records (column names and underline) are to be read as the first two input records.

INTO *expr | DCLvar*

Specifies the position in storage into which input records are to be read.

If the INTO position is specified as an arithmetic expression, *expr*, or a declared variable, *DCLvar*, of numeric or numeric character data type, then a work area must be defined and the numeric value represents a position within the work area.

If the INTO position is a declared variable, *DCLvar*, of character data type, then the input record is a character value that is assigned gets assigned to *DCLvar* following a successful execution of the READ (or CAT) operation.

If a work area has been defined, the default is INTO position 1.

KEY *key*
KEQ

Applicable only to files in which records are organised by a key field within the record data, KEY indicates that a direct read is to be performed for the record containing the key field value specified by *key*.

The *key* value must be an exact match for the key field value in order to successfully read the record. It may be specified as a character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition. If the length of *key* is shorter than the length of the key field, then it is padded with blanks to match the key field length.

If *key* is a character constant, a default of STOPAFT 1 is implied. This may be overridden by specifying an explicit STOPAFT value.

The VSAM access method and the Micro Focus file handler both have in-built support for keyed files (specifically VSAM KSDS and Micro Focus Indexed files) which SELCOPY uses for key lookup. Note that key values in a VSAM KSDS need not be unique if an alternate index is used to input the data set via a READ operation on a PATH DSN. If this is the case, a direct read by key will return the record containing the first occurrence of that key value.

SELCOPY also supports keyed access to z/VM CMS, Windows and Unix-like files in which a key value, located at a fixed position and length within each record, is used to organise records in ascending order of key sequence. To allow this, the KEYPOS parameter must be specified on the READ operation to identify the key field position. The length of the key field is implied by the length of *key*. However, if the length of *key* does not match the length of the key field or another READ KEY operation exists for *fname* where the length of the supplied key value is greater than that of *key*, then KEYLEN must be included to specify the key field length.

Keyed access to Windows and Unix-like files is supported for fixed, variable and undefined record formats, but not for variable formats of type RECFM V2, V3 or MFV. If the record format is variable, then specification of LRECL is mandatory in order to provide a maximum record length with which SELCOPY can validate RDW fields. Since LRECL on a READ operation implies RECFM F, specification of RECFM V would also be necessary.

KEYLEN *int*
KL

Applicable to direct read by key of a z/VM CMS, Windows or Unix-like file, KEYLEN specifies an integer constant value, *int*, which is the length of the key field within the record data.

This parameter is required if the length of the *key* value on the first direct READ operation for *fname* is less than the actual key field length.

See KEY for details of direct read by key.

KEYPOS *int*
KP

Applicable to direct read by key of a z/VM CMS, Windows or Unix-like file, KEYPOS is mandatory and specifies an integer constant value, *int*, which is the position of the key field within the record data.

See KEY for details of direct read by key.

KGE *key*
Like KEY, KGE performs a direct read of a record in a keyed file except that, if no match is found for *key*, the first record containing a key value which is greater than *key* is returned.

KSDS
Applicable to IBM VSAM data sets (z/OS and z/VM CMS) and Micro Focus VSAM files. KSDS identifies the input file as a VSAM KSDS (key sequenced data set) or a Micro Focus indexed sequential (IDXFORMAT 3, 4 or 8) file.

For existing z/OS VSAM data sets, parameter KSDS does not need to be specified as SELCOPY automatically identifies the access method and file organisation of the input file.

For all Micro Focus indexed, relative and record sequential files, parameters VSAM, KSDS, ESDS and RRDS are synonymous and indicate only that input is to be performed via the Micro Focus file handler.

LIST *list_command*
Applicable to z/OS and z/VM CMS, LIST identifies the READ operation as being for SELCOPYi list input.

LIST specifies *list_command*, one of the supported SELCOPYi list commands, that generates a table of listed objects. Optionally specifying SELECT, WHERE and SORT parameter values, selection of list columns, and filtering and sorting of list rows may be performed before the rows are read by the SELCOPY program.

If no *fname* is specified for LIST input, then an *fname* of DEFAULTF is used by default.

The *list_command* SELCOPYi list command syntax may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *list_command* is specified as a declared variable or field definition, then the open (i.e. generation) of the list table is deferred until the READ operation is executed during selection time processing. Therefore, the declared variable names, generated for each input column when the list table is opened, may only be referenced in SELCOPY control statements if *list_command* is a quoted character string.

LRECL *int*
RECSIZE
L
Specifies the maximum input record length for *fname*. If specified on a READ operation without also specifying RECFM, LRECL implies fixed length record format (RECFM F) input.

LRECL should never be specified for input of a z/OS data set, CMS file or IBM VSAM data set. For these types of file input, the maximum input record length is always determined by standard access method processing. If LRECL is specified that does not match the allocated LRECL value, then a file geometry conflict error will be returned.

LRECL is tolerated and ignored if specified on the READ operation for LIST, ODBC table or Micro Focus VSAM input.

For Windows and Unix-like files, including Windows clipboard (CLIP) input, LRECL overrides the maximum record length imposed by the explicit or implied BLKSIZE value. The maximum input record length is determined as follows:

1. The value specified on parameter LRECL.
2. For undefined length record format (RECFM U), the maximum record data length is equal to the size of the input buffer (BLKSIZE) minus the length of the EOL character(s). If EOL is not specified for *fname*, the default EOL character length is 2.
3. For all other record formats (RECFM F, V, VB, V2, V3 and MFV), the maximum record data length is the size of the input buffer (BLKSIZE).

For fixed length record format (RECFM F) input, the maximum record length is the actual length of each input record.

For undefined record format (RECFM U) input, a record exceeding the maximum length will be chopped and treated as though an EOL character was found. The remainder of the record will be treated as a new record.

For variable length record format (RECFM V, VB, V2, V3 and MFV) input, ERROR 537 (RECFM=V BDW/RDW value exceeds BLKSIZE or LRECL) is returned if a record is read with a value in the length field prefix that exceeds the maximum record length value.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

ODBCPASS *pass*
OPASS
PASSWORD
PASSWD
PASS
Applicable to ODBC table input, ODBCPASS specifies the password (*pass*) to be used, together with the associated user name (see USER parameter), when connecting to the data source via ODBC. Note that a user name and password is mandatory for connection to any data source.

pass must be specified as a character constant and its specification will override the prevailing value set by the ODBCPASS environment option.

RAW

For DIR and DIRDATA input of Windows and Unix-like files only, RAW may be specified to prevent SELCOPY formatting of the directory entries.

The content of the physical directory entry as returned by the readdir() C function is returned as the directory record. This may be useful if a directory record is read with an unknown directory type (DIRTYPE).

RBA *byte_num*

Applicable only to VSAM ESDS, Micro Focus record sequential and Windows and Unix-like files.

RBA indicates that a direct read is to be performed for a record at the relative byte address specified by *byte_num* (e.g. RBA 0 is always the relative byte address or the first record). The *byte_num* value may be specified as a numeric constant, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition. If a field definition is used without a TYPE specification, the default is TYPE=P.

If *rba* is a numeric constant, a default of STOPAFT 1 is implied. This may be overridden by specifying an explicit STOPAFT value.

The VSAM access method and Micro Focus file handler have in-built support for ESDS data sets and record sequential files respectively. SELCOPY utilises this functionality to perform record lookup by RBA on these types of file. To successfully perform a direct read by RBA on an ESDS or record sequential file, *byte_num* must be the RBA of the first byte of a record.

SELCOPY supports direct access by RBA on all Windows and Unix-like files except for those with record format represented by RECFM V2, V3 or MFV. Unlike ESDS files, a direct read by RBA will be successful if *byte_num* references any RBA within the file that does not reference addresses beyond position 1 of the last record. If *byte_num* is not the RBA of the first byte of a record, then the record that follows will be returned. Therefore, the value of internal field RBA, which is the RBA of the last record read, may be higher than that specified by *byte_num*.

If the record format is variable (RECFM V or VB), specification of the LRECL parameter is mandatory in order to provide a maximum record length with which SELCOPY can validate RDW fields. Furthermore, if the variable length records contain a key field, as described by KEY, then KEYLEN and KEYPOS parameters may also be specified to further improve RDW field validation.

RDW | **NORDW**

Specifies whether or not the Record Descriptor Word (RDW), which exists before all RECFM V, VB, V2, V3 and MFV (variable length record format) records, is to be presented as part of the input record data and therefore included in the value assigned to variable LRECL.

Beware that, if RDW is specified for a variable length file which is also the prime input object, then, for any WRITE operation, the length of the RDW field is added to the FROM parameter address and deducted from the output data length.

The default is set by environment option, RDW or NORDW. For z/OS, the product default is set by CBLNAME option SRDW=Yes or No. Otherwise, the product default is NORDW.

REC *record_num*

Applicable only to VSAM RRDS, Micro Focus Relative and z/VM CMS files, and also to Windows and Unix-like files processed as RECFM F.

REC indicates that a direct read is to be performed for a record with the record number specified by *record_num*. This record number value may be specified as a numeric constant, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition. If a field definition is used without a TYPE specification, the default is TYPE=P.

If *record_num* is a numeric constant, a default of STOPAFT 1 is implied. This may be overridden by specifying an explicit STOPAFT value.

The VSAM access method and the Micro Focus file handler both have in-built support for numbered record files (specifically VSAM RRDS and Micro Focus Relative files) and standard CMS file input supports direct read by record number. SELCOPY uses these facilities to perform its record lookup.

SELCOPY supports direct access by record number on Windows and Unix-like files only if they contain fixed length records. Parameter LRECL should be used to specify the length of the records.

RECFM **F** | **FB** | **U** | **V** | **VB** | **V2** | **V3** | **MFV**

Specifies the input record format for *fname* as follows:

F	Fixed length records.
FB	Fixed length blocked records.
U	Undefined length records.
V	Variable length records.
VB	Variable length blocked records.
V2	Variable length records with 2-byte length fields.
V3	Variable length File Transfer Protocol (FTP) Block Mode records.
MFV	Variable length Micro Focus record sequential records.

RECFM U may be specified on input of a z/OS data set if record blocks (physical records) are to be processed as single input records. In this case the LRECL value is the allocated BLKSIZE value. Otherwise, RECFM should not be specified for input of a z/OS data set, CMS file or IBM VSAM data set. For these types of file input, record format is always determined by standard access method processing.

RECFM is tolerated and ignored if specified on the READ operation for LIST, ODBC table or Micro Focus VSAM input.

For Windows and Unix-like files including Windows clipboard (CLIP) input, RECFM specifies the format of input data. The default is RECFM U, where records are delimited by end-of-line characters.

See Data Record Format for details of the different RECFM types.

RRDS

Applicable to IBM VSAM data sets (z/OS and z/VM CMS) and Micro Focus VSAM files. RRDS identifies the input file as a VSAM RRDS (relative record data set) or a Micro Focus relative file.

For existing z/OS VSAM data sets, parameter RRDS does not need to be specified as SELCOPY automatically identifies the access method and file organisation of the input file.

For all Micro Focus indexed, relative and record sequential files, parameters VSAM, KSDS, ESDS and RRDS are synonymous and indicate only that input is to be performed via the Micro Focus file handler.

SELECT *select_clause*
SEL

Applicable to SELCOPYi list input only, SELECT defines a subset of the list columns and the order in which they are to be presented for each input list record.

SELECT specifies *select_clause*, a valid SELCOPYi list window SELECT clause. See the ***"SELCOPYi Reference and User Guide"*** for details.

The *select_clause* may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *select_clause* is specified as a declared variable or field definition, then the open (i.e. generation) of the list table is deferred until the READ operation is executed during selection time processing.

SEP *char* | OFF

Applicable only to ODBC table input, SEP specifies a character constant value (*char*), a single character that is used to separate each table column in the input record. The same separator character is used in the generated header records.

By default, *char* is set to "|" (the OR symbol).

SORT *order_by_clause*

Applicable only to SELCOPYi list input and ODBC table input using parameter TABLE, SORT defines the order in which list entries or table rows will be presented to the READ operation.

SORT specifies *order_by_clause* which, for SELCOPYi list input, is valid SELCOPYi list window SORT clause syntax. See the ***"SELCOPYi Reference and User Guide"*** for details.

For ODBC table input, *order_by_clause* is a valid SQL order-by clause which, together with an "ORDER BY" prefix, is used to construct an SQL query statement for the input result table. For details on *order_by_clause* syntax, please refer to SQL reference documentation for the DBMS to which the ODBC connection will be made.

The *order_by_clause* may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *order_by_clause* is specified as a declared variable or field definition, then the object open is deferred until the READ operation is executed during selection time processing.

SORT is also a synonym for SORTDIR.

SORTDIR
SORT

For DIR and DIRDATA input of Windows and Unix-like files only, SORTDIR may be specified to read directory entries ordered by one of the following:

Code	Sort Order
D or T	Date. i.e. The file's last-changed timestamp). Entries are processed in descending (D) or ascending (T) order of date.
X or E	Fileid extension. i.e. The qualifier following the last dot/period. Entries are processed in descending (X) or ascending (E) alphabetical order.
M or N	Fileid name. i.e. All qualifiers preceing the last dot/period. Entries are processed in descending (M) or ascending (N) alphabetical order.
H or P	Fileid path. i.e. The directory path including the fileid name. Entries are processed in descending (H) or ascending (P) alphabetical order.
S or Z	File size. Entries are processed in descending (S) or ascending (Z) order of size.
0 or NO	Unsorted in the order returned by the system.

The default directory input order may be set using the SORTDIR environment option, otherwise the default is 0 (unsorted).

SQL *full_select*

Identifies the READ operation as being for ODBC table input.

For ODBC database input, **SQL** specifies a valid SQL query statement (*full_select*) that will generate a result table. Use of SQL with a query statement is an alternative to specifying TABLE, FMT, WHERE, SORT and UPD parameters. It is also a more flexible method since a result table may be composed of data from more than one table or view. Please refer to SQL SELECT documentation for the DBMS to which the ODBC connection will be made.

If no *fname* is specified for SQL input, then an *fname* of DEFAULTF is used by default.

The *full_select* statement may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *full_select* is specified as a declared variable or field definition, then the object open, connection to the database and execution of the SQL query is deferred until the READ operation is executed during selection time processing. Therefore, the declared variable names, generated for each input column when the table is opened, may only be referenced in SELCOPY control statements if *full_select* is a quoted character string.

SSN *source*

Applicable to ODBC table input, SSN specifies the ODBC data source name (*source*) to which the ODBC connection will be made. Note that specification of an ODBC source name that has been defined to the local system, is mandatory for ODBC table input.

source must be specified as a character constant and its specification will override the prevailing value set by the SSN environment option.

STARTKEY *key*
START

The first execution of a READ operation with STARTKEY will perform a direct read of a record in a keyed file as described for KEY. Subsequent executions of the same READ operation will thereafter perform sequential reads of the same *fname* from the current input position.

Unlike KEY, STOPAFT 1 is not implied if *key* is a constant value.

STARTKGE *key*

READ STARTKGE performs the same processing as READ STARTKEY, except that, if no match is found for *key*, the first record containing a key value which is greater than *key* is returned.

ERROR 544 is returned only if *key* is greater than all the record key field values.

STARTRBA *byte_num*

The first execution of a READ operation with STARTRBA will perform a direct read of a record based on its relative byte address as described for RBA. Subsequent executions of the same READ operation will thereafter perform sequential reads of the same *fname* from the current input position.

Unlike RBA, STOPAFT 1 is not implied if *byte_num* is a constant value.

STARTREC *record_num*

The first execution of a READ operation with STARTREC will perform a direct read of a record based on its record number as described for REC. Subsequent executions of the same READ operation will thereafter perform sequential reads of the same *fname* from the current input position.

Unlike REC, STOPAFT 1 is not implied if *record_num* is a constant value.

STOPAFT *int*
See common parameter STOPAFT for details.

SUBDIR *int*
SUB
For DIR and DIRDATA input of Windows and Unix-like files only, SUBDIR may be specified to identify the number of levels (*int*) of nested sub-directory entries to be included for input. e.g. SUBDIR 1 will only read directory entries for files in the specified (or current) directory, and in all directories whose sub-directory name exists in the specified (or current) directory.

The default number of nested directory levels may be set using the SUBDIR environment option, otherwise the default is 0 (files in the current directory only). If SUBDIR is specified without *int*, a default value of 255 is used.

NOSUB is a synonym for SUBDIR 0.

TABLE *table_name*
TAB
Identifies the READ operation as being for ODBC table input.

For ODBC database input, **TABLE** specifies the name of a table or view (*table_name*) to be read. SELCOPY will construct an SQL query statement using the value specified on TABLE and any values specified for optional parameters FMT, WHERE, SORT and UPD. Use of TABLE and its associated parameters is an alternative to specifying the SQL parameter with an SQL query statement.

If no *fname* is specified for TABLE input, then an *fname* of DEFAULTF is used by default.

The *table_name* value may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *table_name* is specified as a declared variable or field definition, then the object open, connection to the database and execution of the SQL query is deferred until the READ operation is executed during selection time processing. Therefore, the declared variable names, generated for each input column when the table is opened, may only be referenced in SELCOPY control statements if *table_name* is a quoted character string.

TABS *int*
For Windows and Unix-like files which contain tab characters (x'09'), TABS optionally specifies *int*, an integer constant value, which is used to identify tab columns. If TABS is specified without *int*, the default value is 8.

Tab columns are positions within the input record data that are multiples of the value *int*. Where a tab character is found within the input record, then, unless already aligned at a tab column, the data that immediately follows is padded with blanks to the next tab column.

If TABS is specified, then definition of a work area is mandatory, otherwise ERROR 578 is returned.

The default is to process tab characters as data.

TIMES *int*
See common parameter TIMES for details.

UPD
Indicates that *fname* is to be opened for update to allow subsequent execution of SELCOPY UPDATE, INSERT and DELETE operations.

Processing for update is necessary for update-in-place of records in a z/OS data set or library member, and also for update-in-place, delete and insert of records in a VSAM data set.

However, Windows and Unix-like files, z/VM CMS files and Micro Focus VSAM files may be updated without specific update processing. Therefore, specification of UPD is unnecessary for these types of data object.

UPD *update_columns*
Applicable only to ODBC table input using parameter TABLE, UPD identifies the input table columns that are eligible for update by an UPDATE operation.

UPD specifies *update_columns* which is a comma separated list of column names which, together with a "FOR UPDATE OF" prefix, are used to construct an SQL query statement for the input result table. For details on *update_columns* syntax, please refer to SQL reference documentation for the DBMS to which the ODBC connection will be made.

The *update_columns* may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *update_columns* is specified as a declared variable or field definition, then the object open is deferred until the READ operation is executed during selection time processing.

The parameter keyword UPD also applies to file object input.

USER *userid*

Applicable to ODBC table input, USER specifies the user name (*userid*) to be used, together with the associated password (see ODBCPASS parameter), when connecting to the data source via ODBC. Note that a user name and password is mandatory for connection to any data source.

userid must be specified as a character constant and its specification will override the prevailing value set by the USER environment option.

VSAM

Applicable to IBM VSAM data sets (z/OS and z/VM CMS) and Micro Focus VSAM files. VSAM identifies the input file as one of the supported VSAM data set types (KSDS, ESDS or RRDS) or as one of the supported Micro Focus file types (indexed, record sequential or relative).

For z/OS VSAM data sets, parameter VSAM does not need to be specified as SELCOPY automatically identifies the access method and file organisation of the input file.

For all Micro Focus indexed, relative and record sequential files, parameters VSAM, KSDS, ESDS and RRDS are synonymous and indicate only that input is to be performed via the Micro Focus file handler.

WHERE *search_condition*

Applicable only to SELCOPYi list input and ODBC table input using parameter TABLE, WHERE defines a filter for which matching list entries or table rows will be selected for input.

WHERE specifies *search_condition* which, for SELCOPYi list input, is valid SELCOPYi list window WHERE clause syntax. See the ***"SELCOPYi Reference and User Guide"*** for details.

For ODBC table input, *search_condition* is a valid SQL search condition which, together with a "WHERE" prefix, is used to construct an SQL query statement for the input result table. For details on *search_condition* syntax, please refer to SQL reference documentation for the DBMS to which the ODBC connection will be made.

The *search_condition* may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *search_condition* is specified as a declared variable or field definition, then the object open is deferred until the READ operation is executed during selection time processing.

WORKLEN *int*
WORKAREA
WORKA
W

Valid only if specified on the prime input READ operation and only then if WORKLEN is not specified via an OPTION operation.

WORKLEN specifies an integer constant value *int* and indicates that a user work area is to be defined with a number of characters equal to *int*.

The work area will be initialised with blank characters. If a different character is to be used to initialise the work area, then WORKLEN should be specified on an OPTION operation instead, together with a FILL option value.

WTO

Indicates that, at SELCOPY end-of-job, a summary of the number of records read from *fname* will be logged to stderr/SYSOUT. The logged output will have the following format:

```
SELCOPY/xxx r.el       #records=RECTOT  FILE=fname          yyyy/mm/dd hh:MM
```

ODBC Table Read

This feature is currently supported on Microsoft Windows platforms only,

Database table input is supported for a variety of proprietary database products using ODBC (Open Database Connectivity).

ODBC is an API (Application Programming Interface) that SELCOPY uses to access most of the popular databases supporting SQL (Structured Query Language). Each DBMS (DataBase Management System) has its own driver which translates the SQL statements as required.

Specification of a system name, ODBC userid and password is mandatory to identify the ODBC Data Source Name for the required database and to establish the user's access authority to the database tables. Similarly, a user work area must also be defined which has a length which is greater than or equal to the input record (table row) length. If no work area is defined, ERROR 586 is returned.

All column values are presented in a fixed length character data type format, are left adjusted within the column area and padded with blanks to the width of the column area. The width of each column area is the greater of the length of the column name and the length of the column data.

For table columns of fixed or variable character data type, the length of the column data is the maximum defined for the column. For numeric data types, values are converted to character decimal format and have a column data length equal to the minimum length required to display the maximum and minimum values supported by the column definition.

The format and length of data returned by the SQL query for database columns of signed numeric and date data types are as illustrated in the table below. Note that, for all numeric data types except FLOAT and DOUBLE, non-significant zero digits are suppressed. Similarly, negative values are prefixed with "-" (minus), however, positive values are implied and so no sign prefix is included.

Data Type	Display Length	Format
SMALLINT	6	`[-]nnnnnn` `(-32768 to 32767)`
INTEGER	11	`[-]nnnnnnnnnn` `(-2177483648 to 2177483647)`
BIGINT	20	`[-]nnnnnnnnnnnnnnnnnnn` `(-9223372036854775808 to 9223372036854775807)`
DECIMAL(p,s)	p+2	`[-]n*(p-s)[.n*(s)]`
FLOAT	13	`[-]n.nnnnnnE[+/-]nn`
DOUBLE	22	`[-]n.nnnnnnnnnnnnnnE[+/-]nnn`
DECFLOAT(16)	23	`[-]n*(p-s)[.n*(s)]`
DECFLOAT(34)	42	`[-]n*(p-s)[.n*(s)]`
DATE	10	`yyyy-mm-dd`
TIME	8	`hh:MM:ss`
TIMESTAMP	26	`yyyy-mm-dd hh:MM:ss.nnnnnn`

Header Records

Following open for input of a database result table, SELCOPY builds a header line comprising all the selected column names. The column names are padded with blanks to the width of the column data and an optional separator character (default "|") is used to identify the start and end of each column. If sufficient blank space is available within the column width, the length of that column is also included in parentheses. e.g. "`COL_NAME (14) `"

A second header line, with a length equal to that of the first, is generated containing "-" (hyphen) symbols and the same, optional separator characters and serves to underline the column names in the first header line.

The two header lines may be referenced by the variable length field at position FHDR. Furthermore, if the HEADER parameter is specified on the READ operation, the header lines are returned as the first two input records read.

Column Declared Variables

Whether the database table is opened for a READ or WRITE (INSERT) operation, SELCOPY will generate declared variables of fixed length character data type (CHAR), with names that match the column names selected for input or insert. Following each READ operation, values assigned to these variables are updated to reflect the column values of the input row.

The source fields of these declared variables map storage starting at position 1 of the work area. Therefore, if table row data is read into (or inserted from) a position other than position 1 of the work area, an offset must be applied to any reference of the generated declared variable names. e.g. If data is read into position 101, the variable containing the current value of column APILIB, would be referenced as APILIB+101-1.

Note that, if specification of READ operation values for SQL, TABLE, FMT, WHERE, SORT or UPD, are dynamic (i.e. specified as a declared variable name or a field definition), then generated column name declare variables cannot be used. Since use of dynamic values defer the database connection until the statement is executed during SELCOPY's selection time processing, generation of the column name declare variables is delayed and their reference within SELCOPY syntax will trigger an error during control statement analysis.

```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2015/10/15 15:59   PAGE   1
------------------------------------------------------------      ----------------   --------


        ** c:\nbj\slc\ssdb2m07.ctl ***     L=001 --- 2015/10/15 15:59:36  (L05)

          opt worklen=8000    datawidth=80    cblsqlog 'c:\tmp\ssdb2m07.log'

     1.    read  indb2                                                  \
             sql="                                                      \
                   select f.funcname, p.parmno, p.parmname, p.parmtype \
                      from            cblazos.cbl.zzsfunc   f          \
                           inner join cblazos.cbl.zzsparm   p          \
                                       on f.apilib   = p.apilib        \
                                      and f.funcname = p.funcname      \
                     where  f.funcname = 'P2D'                         \
                  order by  p.parmno                                   \
                "                                                       \
                                                                        \
             ssn="DALLAS9"    user="NBJ"    odbcpass="%OPASS%"         \
             sep="|"      header

     2.    print


    INPUT   SEL SEL                                                                                 RECORD
    RECNO   TOT ID.          1         2         3         4         5         6         7         8  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
        1     1   2 FUNCNAME (20)       |PARMNO|PARMNAME (20)       |PARMTYPE (20)       |               70
        2     2   2 --------------------|------|--------------------|--------------------|               70
        3     3   2 P2D                  |1     |Value               |*Packed             |               70
        4     4   2 P2D                  |2     |ValueLength         |Integer             |               70
        5     5   2 P2D                  |3     |Target              |*Char               |               70
        6     6   2 P2D                  |4     |TargetLength        |Integer             |               70
        7     7   2 P2D                  |5     |Flags               |FlagWord            |               70
        8     8   2 P2D                  |6     |Scale               |Integer             |               70
                    ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1             8 READ INDB2      2048    70 F              8        select f.funcname, p.parmno, p.parmname, p.parmtype from ...
    2             8


    ** SELCOPY/WNT 3.30.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                  ** Expiry: 14 Jul 2016 **
```

Figure 16. SELCOPYi ODBC Table Input.

SELCOPYi List Read

On z/OS and z/VM CMS systems, the SELCOPY Product Suite includes the SELCOPYi (interactive environment) product component. SELCOPYi supports command syntax that generates and displays interactive lists of objects including DASD or CMS mini-disk volumes, HFS/ZFS files, data sets and library members.

A list output from SELCOPYi is a table that provides detailed information on each included object. Like a database table read via an ODBC connection, this list table may be read as input to a SELCOPY program using the LIST parameter. The value specified on LIST is the SELCOPYi list command used to generate the list output.

Any of the following SELCOPYi list commands are supported by a SELCOPY READ LIST operation:

Command	Description
AMS	List SYSPRINT output from execution of an IDCAMS command.
CLD	List the first occurrence of each member name in a library concatenation allocated to a specific DDname.
FL	List CMS minidisk files.
FS	List z/OS library members or CMS minidisk files for which a specified search string has been found.
GDG	List cataloged Generation Dataset Groups.
LA	List allocated z/OS and z/VM CMS ddnames.
LAS	List z/OS VSAM associated objects.
LC	List z/OS ICF catalog entries.
LCA	List catalog aliases.
LD	List z/OS cataloged data sets including information retrieved from the VTOC.
LJQ	List z/OS enqueued resources for a specified job name.
LL	List z/OS PDS/PDSE library members.
LPATH	List z/OS HFS/ZFS files.
LQ	List z/OS enqueued resources for a specified queue name.
LSG	List SMS Storage Groups.
LSGV	List SMS Storage Group Volumes.
LV	List z/OS VTOC file entries.
LVOL	List z/OS DASAD volumes or z/VM CMS mini disks.
LX	List z/OS VTOC extents.

Specific list columns may be selected and list entries may be filtered and sorted before being read by SELCOPY. This is achieved using the READ operation SELECT, WHERE and SORT parameters respectively. SELECT, WHERE and SORT values correspond to SELCOPYi command syntax for the SELECT, WHERE and SORT clauses supported by list windows.

A description of the syntax for each of the supported SELCOPYi list commands and for the SELECT, WHERE and SORT clauses is found in the ***"SELCOPYi Reference and User Guide"***.

Like ODBC table read, LIST input has the following characteristics:

1. A work area is required.
2. Column values are presented in a fixed length character data type. Unlike ODBC table input, columns containing numeric values are right adjusted within the column area.
3. **Header records** are generated that are referenced by the FHDR internal field and may be included as record input using the HEADER parameter. Unlike ODBC table input, a blank is used to separate each column and the second header record uses a scale, which has the same format as the last scale line of the SELCOPY list print block, to underline each column name. Note that the width of the scale underline for each column matches the maximum width of the column values and not the column name length.
4. **Column declared variables** are generated for each input list column.

```
SELCOPY/MVS 3.30 at CBL - Bridgend UK (Internal Only)              2015/10/15 15:28   PAGE   1
-------------------------------------------------------            ----------------   --------


        ** NBJ.CTL(SSLST04) ***           L=001 --- 2015/10/15 15:28:46  (NBJ)

          option  worklen=100  datawidth=90

    1.    read  inlst                                                \
            list   "LD NBJ.CBL*.**"                                  \
            select "Entry, Referenced, Org, Trks, Blksz, Lrecl, RecFm" \
            where  "(Referenced >= '2015/01/01') AND (Org = 'PS')"     \
            sort   "Trks D"               header

    2.    print length=90


   INPUT   SEL SEL                                                                                  RECORD
   RECNO   TOT ID.          1         2         3         4         5         6         7         8         9  LENGTH
   -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
       1     1   2 Entry                                    Referenced Org      Trks Blksz Lrecl RecFm          87
       2     2   2 ....,....1....,....2....,....3....,....4.... ....,....1 ... ....,.... ....,  ....,  ....,          87
       3     3   2 NBJ.CBLI.INI.D2015.Z                     2015/01/26 PS       1200 10240 10240 FB            87
       4     4   2 NBJ.CBLATRAC                             2015/09/07 PS        750 27998   512 VB            87
       5     5   2 NBJ.CBLILIBT.D2015288.T143214            2015/10/15 PS         15  1024     0 U             87
       6     6   2 NBJ.CBLI.INI                             2015/10/15 PS          6 32760 32756 VB            87
       7     7   2 NBJ.CBLI330.INI                          2015/04/09 PS          2 32760 32756 VB            87
       8     8   2 NBJ.CBLINST.CBL12075.SELCNAM             2015/08/11 PS          2 23476   256 VB            87
       9     9   2 NBJ.CBL.SMS.LST                          2015/06/02 PS          1  2660   133 FBA           87
      10    10   2 NBJ.CBLI.WINX                            2015/10/15 PS          1 27998   259 VB            87
      11    11   2 NBJ.CBLINST.CBL12075.SELCOPYI.INI        2015/08/11 PS          1 27998   259 VB            87
                   ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1            11 READ INLST              87 F             11       LD NBJ.CBL*.**
    2            11


    ** SELCOPY/MVS 3.30.002  2015/09/17  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 17. SELCOPYi List Data Set Input.

Direct Read

Direct read is the ability to input a record from any location within a file without having to first read records that exist before or after it. Note that, SELCOPY does not support direct read on CAT.

SELCOPY supports direct read for the following input file objects:

- z/OS and z/VM CMS VSAM of KSDS, ESDS and RRDS organisation.
- z/OS MVS data sets and PDS/PDSE library members.
- z/VM CMS files.
- Micro Focus Indexed, Relative and Record Sequential files.
- Windows and Unix-like files.

Supported methods of direct read are by record number (REC), relative byte address (RBA) and key field value (KEY). Specification of READ keyword parameters, REC, RBA, KEY or KGE, with or without a START prefix, will indicate that a direct read of the specified method is to occur.

Note that, not all methods of direct read are supported by the above file objects and, for CMS, Windows and Unix files, certain criteria must also have to be met before particular methods of direct read are possible. Details of the type of file object supported and any pre-requisite criteria that may exist, are documented for each parameter keyword (REC, RBA and KEY) that defines the direct read method to be used.

A START prefix on the direct read keyword indicates that, following the initial direct read, subsequent executions of the same READ operation will perform sequential reads of the same *fname*. The initial direct read will only ever be actioned once for the individual READ operation, even if the file allocated to *fname* changes or is re-opened.

A successful direct read of *fname* will update the current input position within the file so that it points at the matching record. Following the direct read, any sequential read on *fname* that occurs as a result of another READ operation or a START keyword prefix, will input records based on this input position. Therefore, the first record read sequentially following the direct read will be the record after (READ FWD) or preceding (READ BWD) the record that was read directly.

If a START prefix is specified, a direct read which fails to find a match for the requested record will return ERROR 544. Otherwise, a direct read that fails in this way will have the following result:

1. If higher than the current SELCOPY return code value, internal variable RETCODE is updated to the value set by the environment option RC_KEYNF (default 0).

2. Variable LRECL is set to 25 reflecting the length of a generated input record which is returned containing the following text:

   ```
   --- KEY/REC NOT FOUND ---
   ```

   On z/OS and z/VM CMS platforms, this generated record may be suppressed for all executions of SELCOPY via the CBLNAME option SNotFoundMsg=No.

Note that a direct read will not set the end-of-file (EOF) condition. Any sequential reads that are performed subsequently on the same *fname* may set this condition.

Directory Record Read

A directory entry exists, or is generated by SELCOPY, for each z/OS library member, z/VM CMS file or Windows or Unix-like file read with parameter DIR or DIRDATA. Each directory entry is processed as a single input record and may contain meta data (e.g. last changed timestamp, current file size) for the file to which it belongs.

Directory entries are read for files whose fileids match a supplied fileid mask which is assigned to *fname*. For z/OS PDS/PDSE library directory input, the fileid mask may be a library DSN without a member name mask specification and may already be allocated to a ddname referenced as *fname*. The directory entries of multiple PDS/PDSE libraries may be processed using the CAT operation or by allocating a concatenation of libraries to *fname* before SELCOPY is executed.

Lengths of directory records vary but are reflected by the value of the LRECL variable following each directory record read. The format of the directory information is differs depending on the type of file objects to which they refer.

z/OS PDS/PDSE Library Directory Records

Directory records exist for all members in a PDS or PDSE, in blocks of length 256.

The length of the directory entries may vary, due to the presence of user data, up to a maximum of 74 bytes.

Each directory record has a common header in the following format:

Decimal(Hex) Offset	Type	Length	Description
0(0)	Char	8	Member or alias name.
8(8)	Char	3	Relative track and record number. (TTR)
11(B)	Bit string	1	Indicator Byte. `1... ....` Entry is an ALIAS. `.11. ....` Number of TTRs in user data. `...1 1111` Length of user data in half words.

The remainder of the directory record is user data.

The user data for load library directory entries contain many load module attributes which are documented in the IBM publication *"z/OS MVS Program Management: Advanced Facilities."* If updated by SELCOPYi or ISPF, all other libraries have user data formatted as follows:

Decimal(Hex) Offset	Type	Length	Description
12(C)	Binary	1	Version Level.
13(D)	Binary	1	Modification Level.
14(E)	Binary	1	Reserved.
15(F)	Binary	1	Last modified time second.
16(10)	Date	4	Creation Date.
20(14)	Date	6	Last modified date and time.
26(1A)	Binary	2	Current number of records.
28(1C)	Binary	2	Initial number of records.
30(1E)	Binary	2	Modified number of records.
32(20)	Char	8	Last modified by user name.
40(28)	Char	2	Reserved.

If fname has been allocated to library concatenation, then the input record count, assigned to internal variable INCOUNT, will be reset on the first read of each concatenated library. Furthermore, internal field, DSN, will be updated to reference the new library data set name.

Note that, a PDS/PDSE library directory may be processed one block at a time by specifying RECFM U without DIR or DIRDATA, thus reading the directory as a normal sequential file. Standard library directories flag an end-of-file (EOF) condition at the end of directory input so no member data will be read.

```
SELCOPY/MVS 3.30 at CBL - Bridgend UK (Internal Only)              2015/10/15 16:40   PAGE   1
-----------------------------------------------------              ----------------   --------


        ** NBJ.CTL(SSDIR02) ***            L=001 --- 2015/10/15 16:40:14  (NBJ)

          option  worklen=80  datawidth=60 pagewidth=80

     1.    read indir1  dsn='CBL.PRD.SELC330.SZZSSAM1    (ZZSDB*)  '  dir
           cat  indir2  dsn='CBL.PRD.SELC330.SZZSDIST.IPO(ZZSGCFK*)'  dir

     2.    move  8 from fname   to  51
     3.    print  type=b    length=60


    INPUT   SEL SEL                                                          RECORD
    RECNO   TOT ID.          1         2         3         4         5         6  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0  ------

        1     1   3 ZZSDB2B                    N I  NBJ               INDIR1          42
                    EEECCFC401000001013201230 50D0C00DCD4444444444444 4CDCCDF4444
                    9924222008DF1308130F143F9805090052100000000000000 09549910000

        2     2   3 ZZSDB2G                     J    NBJ               INDIR1          42
                    EEECCFC40000010201390132130D0D00DCD4444444444444 4CDCCDF4444
                    992422700D5F1000101F130F22010A0052100000000000000 09549910000

        1     3   3 ZZSGCFKC                     g  NBJ               INDIR2          42
                    EEECCCDC000003450120012810 0B0800DCD4444444444444 4CDCCDF4444
                    99273623037F1F01110F122F480707005210000000000000 09549920000

        2     4   3 ZZSGCFKP                  |   q    JGE               INDIR2          42
                    EEECCCDD000002440120010414090700DCC4444444444444 4CDCCDF4444
                    99273627039F1703110F130F5808030017500000000000000 09549920000
                    ....,....1....,....2....,....3....,....4....,....5....,....6

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1             4 READ INDIR1      256   256 U               2       CBL.PRD.SELC330.SZZSSAM1
                      CAT INDIR2      256   256 U               2       CBL.PRD.SELC330.SZZSDIST.IPO
    2----3        4


    ** SELCOPY/MVS 3.30.002  2015/09/17  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                             ** Expiry: 14 Jul 2016 **
```

Figure 18. SELCOPY Directory Input for z/OS Library Members.

z/VM CMS File Directory Records

Directory records are generated by SELCOPY from FST data returned by read of CMS files with DIR or DIRDATA. The length of the directory entries vary between 88 and 103 bytes, depending on the length of the file name, type and mode, and are formatted as follows.

Decimal(Hex) Offset	Type	Length	Description
0(0)	Char	8	File name.
8(8)	Char	1	Blank.
9(9)	Char	8	File type.
17(11)	Char	1	Blank.
18(12)	Char	2	File mode.
20(14)	Char	2	Blanks.
22(16)	Char	10	Last modified date. (yyyy/mm/dd)
32(20)	Char	1	Blank.
33(21)	Char	8	Last modified time. (HH:MM:SS)
41(29)	Char	1	Blank.
42(2A)	Char	8	Number of records. Right adjusted with comma punctuation.
50(32)	Char	1	Blank.
51(33)	Char	8	LRECL value. Right adjusted with comma punctuation.
59(3B)	Char	1	Blank.
60(3C)	Char	2	RECFM.
62(3E)	Char	1	Blank.
63(3F)	Char	7	Number of blocks. Right adjusted with comma punctuation.
70(46)	Char	1	Blank.
71(47)	Char	6	Volume label.
77(4D)	Char	1	Blank.
78(4E)	Char	3	Link authorisation. (R/W or R/O)
81(51)	Char	2	Blanks.
83(53)	Char	Variable (max 20)	Fileid with dot/period separators. (fn.ft.fm)

```
SELCOPY/CMS 3.30 at CBL - Bridgend UK (Internal Only)              2015/10/15 14:37   PAGE   1
-----------------------------------------------------              ----------------   --------


        ** SSDIR02 CTL A ***               L=002 --- 2015/10/15 14:37:55  (VMNBJ)

     1.    read    'S%%%%% MODULE *'    dir
     2.    print



    INPUT   SEL SEL                                                                                                1  RECORD
    RECNO   TOT ID.          1         2         3         4         5         6         7         8         9         0  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
        1     1   2 SABBEY   MODULE   A1  2007/09/25 10:51:12        7       80 V        1 PC05A  R/W  SABBEY.MODULE.A1       99
        2     2   2 SELUPD   MODULE   G1  2007/10/12 14:03:20        4   65,535 V       17 PC00G  R/O  SELUPD.MODULE.G1       99
        3     3   2 S310AZ   MODULE   I1  2012/03/08 15:23:34        4   65,535 V       32 PC00I  R/O  S310AZ.MODULE.I1       99
        4     4   2 SALIPL   MODULE   S2  2012/04/16 15:12:19        3   60,992 V       16 MNT190 R/O  SALIPL.MODULE.S2       99
        5     5   2 SAVEFD   MODULE   S2  2011/09/27 14:10:18        2    3,752 V        1 MNT190 R/O  SAVEFD.MODULE.S2       99
        6     6   2 SEGGEN   MODULE   S2  2011/09/27 14:10:19        3   29,800 V        8 MNT190 R/O  SEGGEN.MODULE.S2       99
        7     7   2 SETKEY   MODULE   S2  2011/09/27 14:10:19        3    1,056 V        1 MNT190 R/O  SETKEY.MODULE.S2       99
        8     8   2 SETPRT   MODULE   S2  2011/09/27 14:10:20        2    6,784 V        2 MNT190 R/O  SETPRT.MODULE.S2       99
        9     9   2 SHRLDR   MODULE   S2  2011/09/27 14:10:21        4    2,552 V        1 MNT190 R/O  SHRLDR.MODULE.S2       99
       10    10   2 SDB98P   MODULE   V1  2001/06/25 13:49:58        4   22,272 V        6 VM02V  R/O  SDB98P.MODULE.V1       99
                    ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,....0

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1            10 READ S%%%%%     2048  2048 U             10       S%%%%%.MODULE.*
    2            10


    ** SELCOPY/CMS 3.30.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                  ** Expiry: 14 Jul 2016 **
```

Figure 19. SELCOPY Directory Input for z/VM CMS Files.

Windows and Unix-like File Directory Records

Directory records are generated by SELCOPY from readdir() dirent output for Windows and Unix-like files read with DIR or DIRDATA, and may be of any length depending on the length of the full fileid file path.

The generated list of directories are filtered based on the DIRTYPE and SUBDIR parameters, and are sorted based on the SORTDIR parameter.

DIRTYPE selects entries based on file attributes, and may also be used to include directory records that are not associated with a file (e.g. sub-directories, volume id and the working (.) and parent (..) directories).

SUBDIR allows entries to be included for files that match the fileid mask and exist in a sub-directory of the working directory. The number of nested sub-directory levels eligible for directory input is determined by the value specified for SUBDIR (default). If SUBDIR is not specified, entries for files in sub-directories are not included (equivalent to SUBDIR 0).

SORTDIR may be specified to process directory records in a specified order: by fileid name, path, extension, size or timestamp. By default, directory records are returned unsorted (SORTDIR NO or 0). i.e. in the order returned from the operating system.

Unless RAW is specified, the format of the directory records is as follow:

Decimal(Hex) Offset	Type	Length	Description
0(0)	Char	10	Last modified date. (yyyy/mm/dd)
10(A)	Char	1	Blank.
11(B)	Char	8	Last modified time. (HH:MM:SS)
19(13)	Char	2	Blanks.
21(15)	Char	6	**Unix-like files only**: File type and permision flags as described by the ***st_mode*** field of the *stat* struct. For details, see documentation for *st_mode* in the ***stat(2)*** man page.
21(15)	Char	7	**Windows files only**: 3 blank characters followed by 1 character each for applicable file attributes in the following order: "a" (amended), "r" (read-only), "h" (hidden) and "s" (system).
28(1C)	Char	1	Blank.
29(1D)	Char	8	Fileid name without the extension. "*" (asterisk) is displayed as the 8th character if the length is greater than 8.
37(25)	Char	1	Dot/period.
38(26)	Char	3	Fileid extension (the fileid qualifier following the last dot/period). "*" (asterisk) is displayed as the 3rd character if the length is greater than 3.
41(29)	Char	1	Blank.
42(2A)	Char	15	Number of bytes file size. Right adjusted with comma punctuation.
57(39)	Char	2	Blanks.
59(3B)	Char	Variable	The complete fileid including the file path.

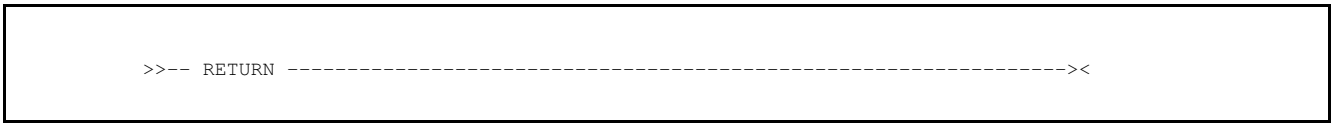
```
SELCOPY/LNX 3.20 at CBL - Bridgend UK (Internal Only)                                                    2015/10/15 15:02   PAGE   1
-----------------------------------------------------                                                    ----------------   --------


          option pagewidth=128

     1.    read  '/home/nbj/*'   dir  dirtype=DNY  subdir=1 sortdir=D
     2.    print  stopaft=10


    INPUT   SEL SEL                                                                                                  1  RECORD
    RECNO   TOT ID.          1         2         3         4         5         6         7         8         9         0  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
        1     1   2 2015/10/15 15:02:16  100644  ssdir01 .lst              0  /home/nbj/ssdir01.lst                          80
        2     2   2 2015/10/15 15:02:14  100600  .viminfo.             3,864  /home/nbj/.viminfo                             77
        3     3   2 2015/10/15 15:02:13  100644  ctlfile .               104  /home/nbj/ctlfile                              76
        4     4   2 2015/10/15 15:02:13  100644  ssdir01 .ctl            104  /home/nbj/ssdir01.ctl                          80
        5     5   2 2015/10/14 17:21:00  100600  .bash_h*.            10,832  /home/nbj/.bash_history                        82
        6     6   2 2015/10/14 09:38:46   40700  88b63d3*.             4,096  /home/nbj/.pulse/88b63d335ff572654632a034      116
      +100          0000006f-runtime
        7     7   2 2015/10/14 09:38:26   40700  .gconf  .             4,096  /home/nbj/.gconf                               75
        8     8   2 2015/10/14 09:38:26   40700  .gconfd .             4,096  /home/nbj/.gconfd                              76
        9     9   2 2015/10/14 09:38:26  100600  .xsessi*.             2,900  /home/nbj/.xsession-errors                     85
       10    10   2 2015/10/14 09:38:26  100700  saved_s*.            52,527  /home/nbj/.gconfd/saved_state                  88
                    ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,....0


SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1            10 READ *          2048  2046 U             10       /home/nbj/*
                                 *EOF*NOT*REACHED*
    2            10


***WARNING***                   4 = RETURN CODE FROM SELCOPY

    ** SELCOPY/LNX 3.20.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                    ** Expiry: 14 Jul 2016 **
```

Figure 20. SELCOPY Directory Input for Windows and Unix-like Files.

RETURN

Return processing from a SELCOPY sub-routine back to the caller.

Syntax:



Synonyms:

RETURN	RET

Description:

The RETURN operation must be used to exit a SELCOPY internal sub-routine.

On execution of RETURN, processing control is passed back to the next executable statement following the DO operation that called the sub-routine. This operation may be in the main processing loop or another sub-routine. It may also exist in the same sub-routine if it has been called recursively. e.g. The following will recursively call a sub-routine to calculate a factorial value.

```
  DECLARE           FVAL  BIN    INI=1    FORMAT='Z,ZZZ,ZZ9'

       @WVAL  =     10
  DO  FACTORIAL   @WVAL   FORMAT='Z,ZZZ,ZZ9'
  PRINT  '10! = '  FVAL                        * Print result.
  EOJ

==FACTORIAL:==     XX                          * Sub-routine with 1 input value.
       @WVAL  =    XX
  IF   @WVAL  =    0          !THEN RETURN     * Return when XX=0
  MULT             FVAL   BY  @WVAL
  DO  FACTORIAL   @WVAL-1 FORMAT='Z,ZZZ,ZZ9'   * Recursive call.
=RETURN=                                       * Return at end of sub-routine.
```

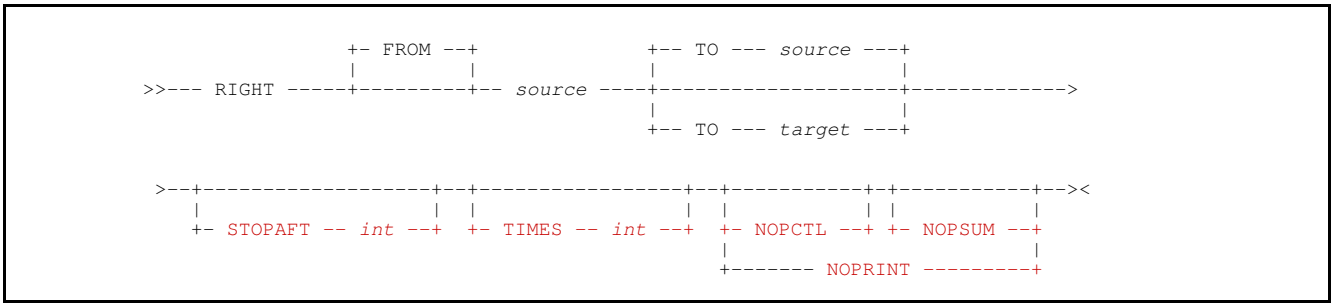
SELCOPY internal sub-routines are not private procedures and so all variable and field definitions are exposed to the routine. Any changes made to variable and field definition values (including internal variable RETCODE) persist when control is passed back to the calling routine or main processing loop.

Because of this RETURN does not support any parameter arguments that assign a return value.

RIGHT

Right adjust text in a character source value.

Syntax:



Synonyms:

RIGHT	ADJR

Description:

The text represented by the *source* character value is right adjusted and assigned to the *target* value. The *target* location may overlap storage at the *source* value location and if *target* is not specified, then the *source* value is updated.

The source text value is right adjusted in an area of storage containing all blank characters and of length equal to the *target* value field area if specified, otherwise the length of the *source* value field area.

```
DECLARE  PAGE   CHAR(20)  INI='Page'
DECLARE  PNO     BIN      INI=2   FORMAT=999
RIGHT    PAGE                     *          ....,....1....,....2....,
PRINT    PAGE  ' '  PNO            * Prints: "               Page 002"
```

Multiple consecutive blank characters that exist between non-blank characters in the source value text, are condensed to a single blank character in the right adjusted value. e.g.

```
POS     101 = "Hello    World "        *          ....,....1....,....2
RIGHT   15 AT 101   TO  POS 1, 20      * To get: "         Hello World"
```

Parameters:

FROM *source*
References the character text value to be right adjusted. *source* may be specified as a declared variable of character data type, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition.

If TO *target* is specified, *source* may also be specified as a quoted or hex character constant.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *target* is implied. Otherwise, ERROR 69 is returned.

TO *target*
Identifies the declared variable or Type 1, 2 or 3 field definition that is the target of the operation. If *source* is a constant value, then TO *target* is mandatory.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *source* is implied.

Default is TO *source*.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

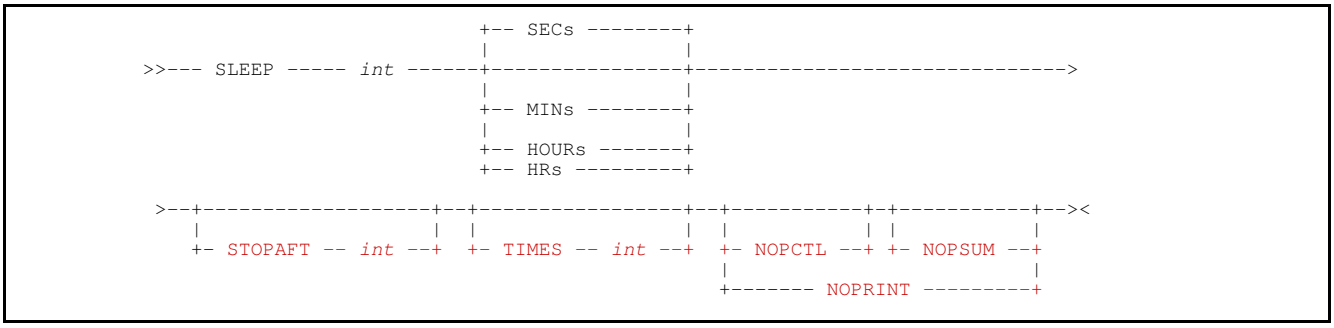
STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of RIGHT for the same source and target values is unnecessary and so TIMES should never be used on the RIGHT operation. See common parameter TIMES for details.

SLEEP

Pause program execution for a period of time.

Syntax:



Description:

The SLEEP operation is supported in Windows and POSIX environments only.

During selection time processing, the SLEEP operation provides the capability of waiting on the system timer before proceeding with execution of the next statement. This may be useful for a job that periodically checks for an event. e.g. receipt of an email or completion of another process.

The duration of time for which processing is paused, may be specified in seconds, minutes or hours.

Parameters:

int **SECS** | **MINS** | **HOURS** | **HRS**
A positive decimal integer constant (*int*) specifying the number of seconds (SECS), minutes (MINS) or hours (HOURS or HRS) to pause. Default is *int* SECS.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Example:

The following example illustrates how the SLEEP operation may be used to pause processing for 11 seconds before testing a condition that changes with elapsed time. If the condition is true, the processing repeats until the condition tests false.

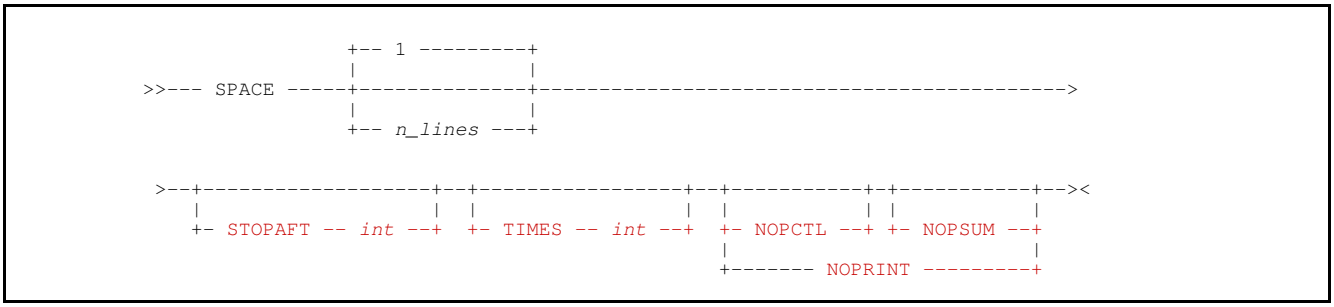
```
** g:\cc\slc\ctl\SSSLEEP1 **       L=001 --- 2000/12/28 17:57:22  (P21)
option  worklen=222  dw=60    noban
  do prtdat
  move 4 at DATE+64   to 201     * Save original "Secs since midnight" value.

=waitloop=
  sleep  11 secs
  do some-rtn        * (Perform some check for an event.)
  if .......        * (  If event not here yet.        )
    then goto waitloop

=gotit=                          * Event has occurred.
  cvdate   NOW   to datecb       * Refresh the POS DATE Control Block with the time NOW.
  do prtdat
*             ....,....1....,....2...
  pos  101 = 'Runtime = nnnn seconds.'
  sub    4 at 201 type=b   from 4 at DATE+64 type=b    into  4 at 205 type=b
  cvbc   4 at 205          to 111   format  zzz9
  print  from 101  length=60
  eoj

=prtdat=
  move  19 at DATE-02      to   1
  cvbc   4 at DATE+64      to  21   format 'zzz,zz9'
  pos   29 = 'secs since midnight.'
  print            length=60
=return=
```

Figure 21. SLEEP operation sample usage.

SPACE

Write one or more blank lines to the SELCOPY list output.

Syntax:



Description:

Execution of the SPACE operation will skip lines in the SELCOPY list output so that, when the next line of text is written to the same page of the list output, blank lines occur before it. If SPACE follows execution of a PRINT operation, the blank line(s) will occur within the print block.

Lines at the start of a page will not be skipped by a SPACE operation. If the next output occurs at the start of a new page, following any page header lines, then a SPACE operation will have no effect. If lines are to be skipped at the start of a page then a value must be assigned to internal variable LINE. (e.g. LINE=6 will write the next line of text at line 6)

Furthermore, if the number of lines to be skipped by a SPACE operation exceeds the number of lines remaining on the current page, then a new page will be started immediately following the last output text line. No lines will be skipped at the start of that new page to account for the difference.

Parameters:

n_lines
: Indicating the number of lines to be skipped, *n_lines* may only be specified as a positive, non-zero integer constant value.

  Default is 1.

NOPCTL, NOPSUM, NOPRINT
: See common parameters NOPCTL, NOPSUM, NOPRINT for details.

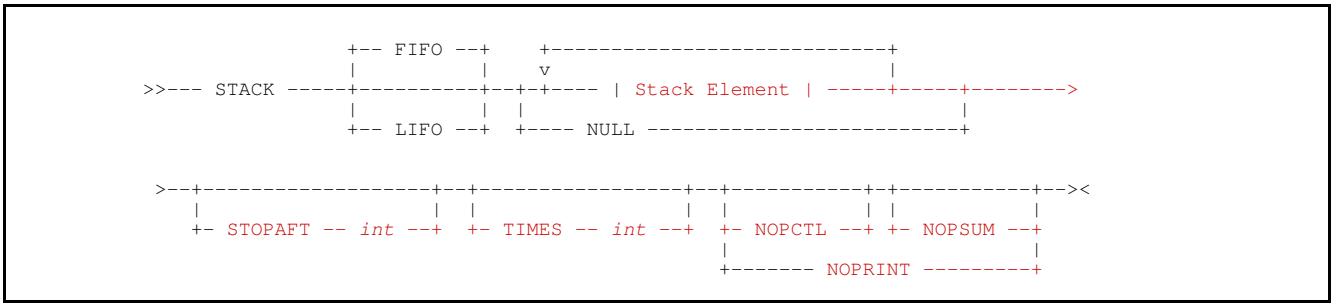
STOPAFT *int*
: See common parameter STOPAFT for details.

TIMES *int*
: See common parameter TIMES for details.

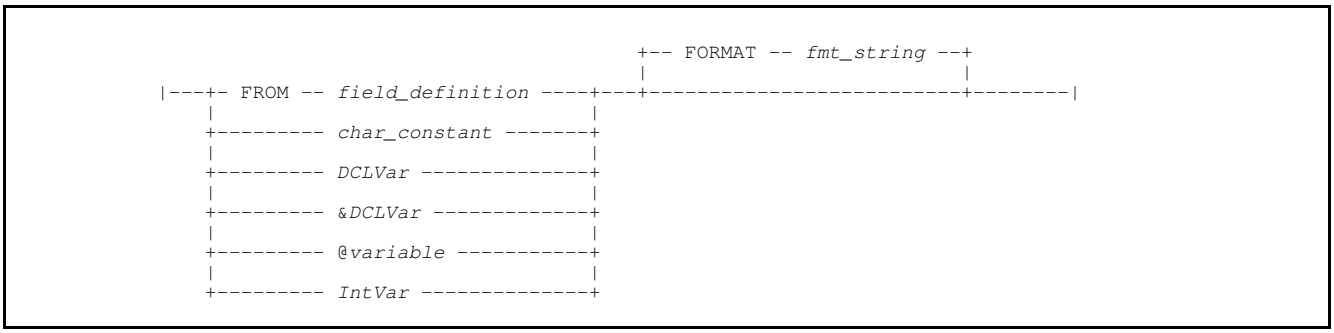
STACK

Add an entry to the z/VM CMS program stack.

Syntax:



Stack Element:



Description:

Supported only in z/VM CMS environments, the STACK operation will add a line of text or a null line to the CMS program stack.

By default, a line is stacked FIFO (First-In, First-Out) so that it is added following any previously stacked lines of text. If parameter LIFO (Last-In, First-Out) is specified, the line will be added and subsequently processed ahead of any previously stacked lines of text.

Note that the length of the line of text to be stacked must not exceed 255 characters. This is a limitation of the CMS program stack. The system return code value set by the execution of STACK is assigned to the internal variable RETSYS.

If the READ CARD operation is used, then stacked lines may be read as input to the same SELCOPY program. On completion of the SELCOPY program execution, control is passed back to CMS and any line left in the program stack will be treated by CMS as a command.

Beware that, if the SELCOPY program has been started from an application that has its own command processor (e.g. SELCOPYi or XEDIT), then the lines remaining in the stack when the program finishes will be interpreted as application commands. If a stacked command is to be executed by CMS and the SELCOPY program will always be executed from one of these types of application, then prefixing the stacked command with CMS will ensure it is passed to CMS for processing. e.g.

```
STACK   "CMS COPY   * * A   = = B   (OLDD"
```

Parameters:

`Stack Element`
Specifies an element that constitutes a portion of the stacked line of text.

Multiple stack elements are concatenated, in the order in which they are specified, to construct the complete line of text.

Stack element values that are of numeric or character numeric data type are automatically converted to decimal character display format. Use &*DCLVar* to obtain the unformatted value of a numeric *DCLVar*.

Each stack element may be specified in one of the following formats:

`FROM` *`field_definition`*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type.

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL. However, if parameter FORMAT is used to convert the field data to printable hex, specification of a length is mandatory.

Although not necessary, keyword FROM may also be used before an stack element specified as a *char_constant* or a *DCLVar* of character data type.

char_constant
A quoted or hexadecimal character constant.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
The name of an @variable that has a non-null value.
If *@variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
The name of a SELCOPY internal variable.

FORMAT *fmt_string*
FMAT
FMT
FORMAT specifies *fmt_string*, a format string used as the character display template for data specified by the stack element.

The source data in a stack element of numeric data type is automatically converted to displayable character format (using a CVxC operation). The length of the stack element is determined by *fmt_string*, not the length of the stack element source.

For stack elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For stack elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the stack element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For a stack element specified as an *@variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
`'SS,SSS,SSS,SS9'`	Used for *@variable*, *IntVar* and *DCLVar* representing an integer value.
`'SS,SSS,SSS,SS9.9999'`	Used for *DCLVar* representing a rational value.

FIFO | **LIFO**
Stack the line as FIFO (First-In, First-Out) following existing stacked lines, or LIFO (Last-In, First-Out) in front of existing stacked lines.

Default is FIFO.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

NULL
EOF
Stacks a null line. When read as input from the stack, a null line will indicate end-of-file.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Example:

The following sample output is from a SELCOPY program that reads file list entries which match a CMS fileid mask with file type "CTL" and file mode "A". The file name is supplied as a parameter to the program and can include wild card characters (in this case "XS*").

The example demonstrates use of a conditional STACK operation to execute a CMS ERASE command for a temporary output file if no input list entries exist.

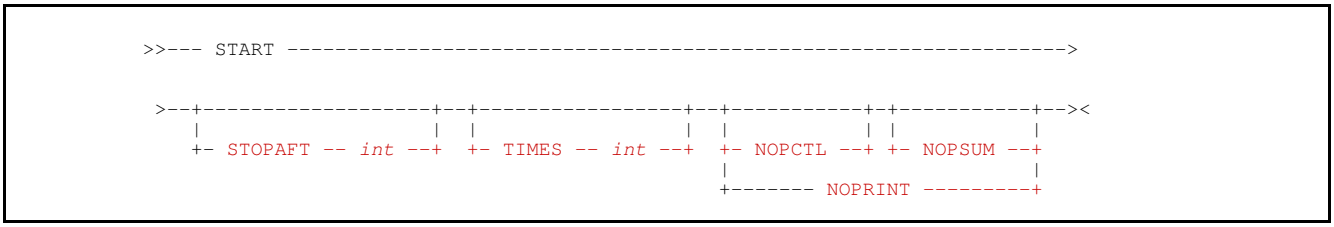
```
SELCOPY/CMS 3.30 at CBL - Bridgend UK (Internal Only)              2016/01/06 12:14   PAGE   1
-------------------------------------------------------            ----------------   --------


         EQU %1   XS*


           OPT  DATAWIDTH=50

     1.    DO        INIT_OFILE   STOPAFT=1     * Initialise Output file

     2.    READ      INLST   LIST='FL %1 CTL A'

           IF EOF
     3.      THEN DO CLEAN_UP
     4.      THEN EOJ


     5.    WRITE  OWORK  DSN='SSSTAC01 TEMP B'
     6.    GOTO GET


         ==INIT_OFILE==
           ----------
     7.    WRITE  OWORK  'Output from SSSTAC01 CTL ('  FROM 19 AT DATE-2  ')'
     8.    WRITE  OWORK  ' '
     9.   =RETURN=


         ==CLEAN_UP==
           --------
           IF INCOUNT = 0
    10.      THEN PLOG   'No entries found for:   %1 CTL A'
    11.      THEN PLOG   'Erasing report file SSSTAC01 TEMP B.'
    12.      THEN STACK  'ERASE SSSTAC01 TEMP B'

    13.   =RETURN=


    INPUT   SEL SEL                                                      RECORD
    RECNO   TOT ID.          1         2         3         4         5  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0  ------
        0     1  10 No entries found for:   XS* CTL A                      80
        0     1  11 Erasing report file SSSTAC01 TEMP B.                   80
                    ....,....1....,....2....,....3....,....4....,....5


SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1             1
    2             0 READ INLST             105 F               0       FL XS* CTL A
    3----4        1
    5             0 WR   OWORK      1032    46 V               2       SSSTAC01.TEMP.B1
    6             0

                    =INIT_OFILE=
    7             1 WR   OWORK      1032    46 V               2       SSSTAC01.TEMP.B1
    8             1 WR   OWORK      1032    46 V               2       SSSTAC01.TEMP.B1
    9             1 =ret=

                    =CLEAN_UP=
   10---13        1 =ret=


    ** SELCOPY/CMS 3.30.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                  ** Expiry: 14 Jul 2016 **
```

Figure 22. STACK a CMS command.

START

Start selection time processing which has been suspended.

Syntax:



Synonyms:

START	FILE=START

Description:

Execution of the START operation will place SELCOPY's selection time processing in a started state.

In started state, selection time processing will execute all condition operations and executable statement selections as dictated by SELCOPY's standard logic flow. This is the default state for SELCOPY program execution.

The START operation is required to return processing to the started state having been placed in suspended state by the SUSP operation. Note that the presence of a START operation within the control statements will also force selection time processing to begin in the suspended state.

Execution of START, when processing is already in the started state, executes a null operation and the selection count is unchanged. Otherwise, selection processing activity proceeds from the first statement following the START operation.

When processing in suspended state, SELCOPY will only execute prime input READ operations and IF operations for which a conditional START operation is specified. Therefore, specifying START as an unconditional operation will have no effect.

Parameters:

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of START is unnecessary and so TIMES should never be used on a START operation. See common parameter TIMES for details.

Example:

The following example reads a file containing report information for each branch office in an organisation and prints only the information for branch offices referenced as H149. The input file is organised in ascending order of branch office number.

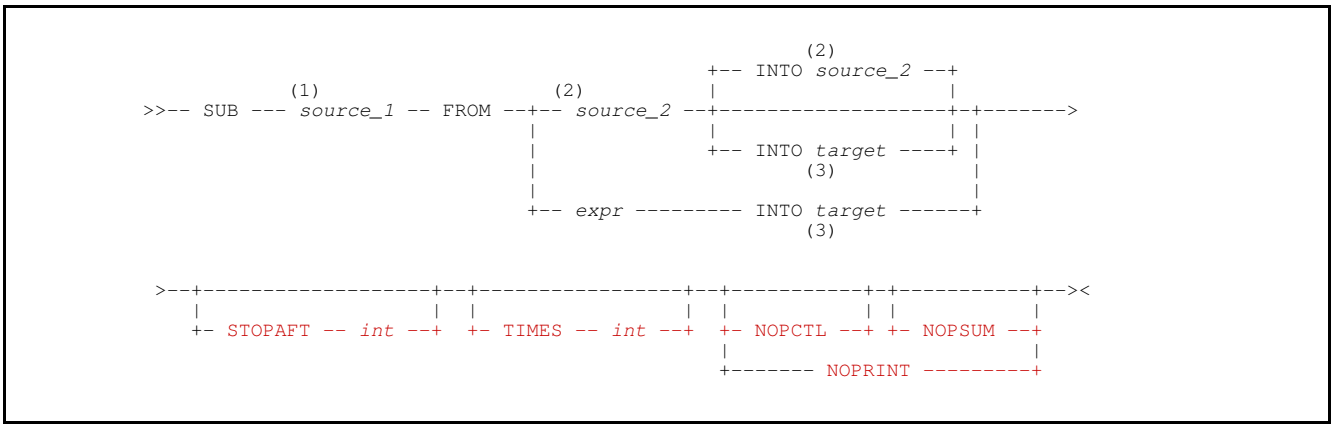
Because the START operation exists, processing begins in suspended state and is placed in the started state when the header record for branch H149 is read. Having completed processing for branch office H149, end-of-job processing is started.

```
READ   MASTER                   * Report file for each branch office.

IF POS 1 = 'H150'  !THEN EOJ    * Terminate after branch H149.
IF POS 1 = 'H149'  !THEN START  * Header record for branch H149.

PRINT                           * Only if in START state.
```

SUB

Performs an arithmetic subtraction of two, signed numeric source values, the result of which is assigned to a target variable or stored in a target field.

Syntax:



Syntax Notes:

(1) ***source_1*** must be of numeric or character numeric data type and may be specified as a constant, an arithmetic expression (*expr*), a declared variable or as a field definition of type 2 (*field_p1p2*) or type 3 (*field_nATp*).

(2) ***source_2*** must be of numeric or character numeric data type and may be specified as a declared variable or as a field definition of type 2 (*field_p1p2*) or type 3 (*field_nATp*).
If INTO *target* is specified, *source_2* may also be specified as a constant value or an arithmetic expression (*expr*).

(3) ***target*** may be specified as a declared variable or as a field definition of type 2 (*field_p1p2*), type 3 (*field_nATp*) or type 4 (*field_pFMT*).

Description:

Variables and field definitions that constitute the source values and target of the SUB operation may be of different numeric or character numeric data types.

When the source and target values are of different data types and/or if an arithmetic expression or constant value is specified as a source field, then values are converted to a common numeric data type prior to performing the subtraction. The selected common data type will have a precision and scale which is sufficient to manage the maximum precision and scale of the resultant value.

Any rational number having a fraction value, will be stored internally as a double precision (8-byte) floating point number. Therefore, all fields in the operation will be converted to double precision floating point data type and then floating point arithmetic used.

If a target or source value is expressed as a field definition without a TYPE parameter, the default data type attributed to that field will be the same as the first field definition within the command syntax to have TYPE specified. If no such field definition exists, the default data type is packed decimal integer (TYPE=P). e.g.

```
SUB  4 AT 1 TYPE=C  FROM 4 AT 6 TYPE=B   INTO 4 AT 11          * Default TYPE=C
SUB  4 AT 1 TYPE=B  FROM 4 AT 6 TYPE=C   INTO 4 AT 11          * Default TYPE=B
SUB  4 AT 1         FROM 4 AT 6          INTO 4 AT 11 TYPE=C   * Default TYPE=C
SUB  4 AT 1         FROM 4 AT 6          INTO 4 AT 11          * Default TYPE=P
```

Rounding will occur as required on the fractional part of the resultant value based on the number of fractional digits (scale) of the target definition. e.g. If the target variable or field has no scale, the result will be rounded to the nearest integer value by adding 0.5 and dropping the fractional portion.

Parameters:

source_1
Represents a rational numeric value (subtrahend) that is to be subtracted from the value specified by *source_2*.

FROM *source_2*
Represents a rational numeric value (minuend) to which the value specified by *source_1* will be subtracted.

If no *target* is specified, then *source_2* is also the target of the operation. If this is the case, *source_2* must be a declared variable or a field definition.

expr
Represents a numeric or character numeric constant, or an arithmetic expression to be used as the minuend value.

Note that *source_1* may also be specified in this way. The distinction is made for *expr* and *source_2* because *source_2* must be a valid target if INTO is not specified. Therefore, if *expr* is used, INTO *target* is mandatory.

INTO *target*
Identifies the declared variable or field definition that is the target of the operation and to which the resultant value (difference) is assigned. If the second source value is *expr*, then *target* is mandatory.

If required, data conversion and decimal rounding will be performed on the resultant value before it is assigned to *target*.

Default is *source_2* specified on the FROM parameter.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Examples:

```
  DECLARE  GRAVITY  FLOAT   INI=9.80665         * Declared variable.

  EQU      OREC    1001                         * Output record area.

  OPTION   WORKLEN 2000                         * Work area buffer.

  @VAR        =    8 AT  23 TYPE=P              * Assign @variable value.

  SUB  18       FROM   4 AT  51 TYPE=B
  SUB  2.1      FROM   8 AT  23 TYPE=P      INTO  8 AT 161 TYPE=F BIN
  SUB  GRAVITY  FROM   8 AT 161 TYPE=F BIN
  SUB  @VAR     FROM   GRAVITY              INTO  OREC+12   FMT='S,SS9.99'
```

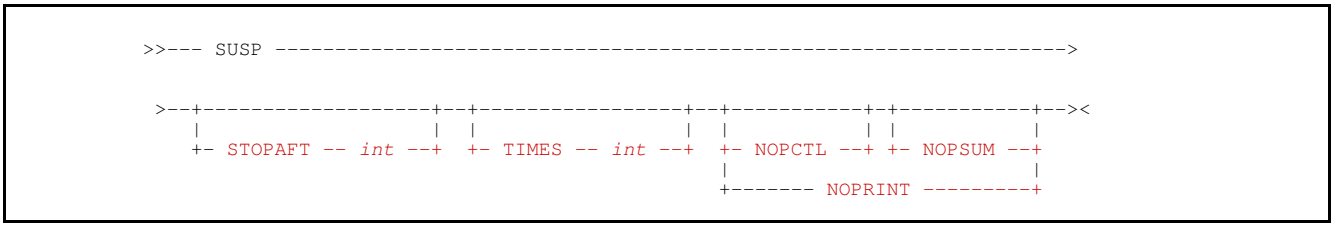
Return Codes:

0	Successful completion.
8	One of the following conditions has occurred: 1. *target* has defaulted to *source_2*, which is of data type binary or floating point, and arithmetic overflow has occurred. i.e. A positive value has been subtracted from a negative value *source_2* and the result is positive or a negative value has been subtracted from a positive value in *source_2* and the result is negative. 2. The precision of *target* is not sufficient to contain the resultant value. Truncation has occurred with the loss of significant digits. 3. The first byte of a source or target field definition is within the work area but the last byte is located beyond the end of the work area buffer. The length of the field is reduced so that the last byte of the field is the last byte of the work area buffer. 4. At least one source value is treated as being of packed decimal data type but the source data is invalid packed decimal data.

SUSP

Suspend selection time processing.

Syntax:



Synonyms:

SUSP	FILE=SUSP

Description:

Execution of the SUSP operation will place SELCOPY's selection time processing in a suspended state.

For large prime input data objects in which only a small subset of the records/rows are to be processed, a SELCOPY program's efficiency may be enhanced by suspending processing of the program's statements for all unrequired records/rows. For LIST and ODBC table input, unrequired rows can be omitted using row filtering techniques. Therefore, SUSP is mainly used for file input where record filtering is not supported by the I/O access methods and so must be performed by the SELCOPY program itself.

Whilst in suspended state, selection time processing will only execute the following:

1. Statements containing an IF/AND/OR condition operation for which a conditional START operation exists. e.g.

   ```
   IF POS 11 = 'E15'
   OR POS 11 = 'H12'
     THEN START
   ```

2. Statements containing a READ operation for the prime input object.

   Records/rows read from this object are not passed from the input buffer to a user defined work area or a character declared variable source field. Therefore, if SUSP is used, specification of parameter INTO on the prime input READ operation is invalid and will return an error.

Processing of all other condition operations and executable statements will be bypassed until a START operation is executed.

Selection time processing is automatically placed in a suspended state if a START operation exists within the control statements. Therefore, the SUSP operation is only required if processing is to be conditionally returned to a suspended state following execution of a START operation. See the example below.

Parameters:

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of SUSP is unnecessary and so TIMES should never be used on a SUSP operation. See common parameter TIMES for details.

Example:

The following example reads a file containing report information for each branch office in an organisation and prints only the information for branch offices referenced as H149, H153 and H206. The input file is organised in ascending order of branch office number.

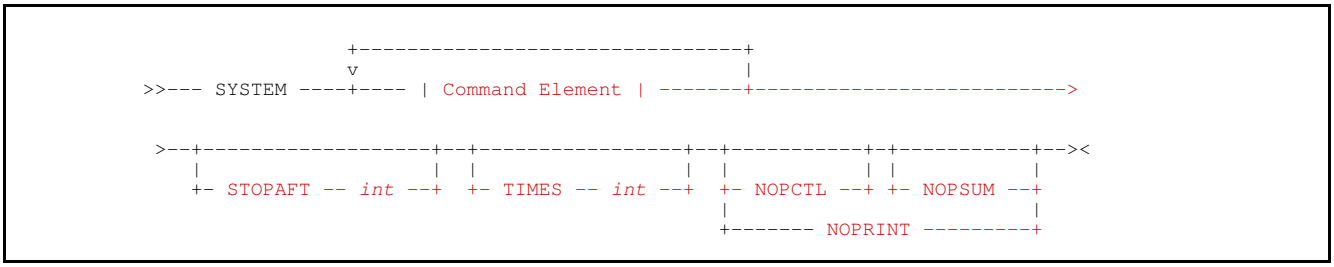
Because the START operation exists, processing begins in suspended state and is placed in the started state when the header record for branches H149, H153 and H206 are read. For all other branch offices in between, processing is returned to suspended state.

```
READ   MASTER                  * Report file for each branch office.

IF      POS 1  = 'H'           * If the start of new branch office output.
THEN IF POS 2  > '206'         * If branch offices following 206.
       THEN EOJ                * Terminate after branch 206.
       ELSE SUSP               * Suspend processing all branches.

IF POS 1 = 'H149'              * Header record for 149 branch.
OR POS 1 = 'H153'              * Header record for 153 branch.
OR POS 1 = 'H206'              * Header record for 206 branch.
  THEN START                   * Activate required branches.

PRINT                          * Only if in START state.
```

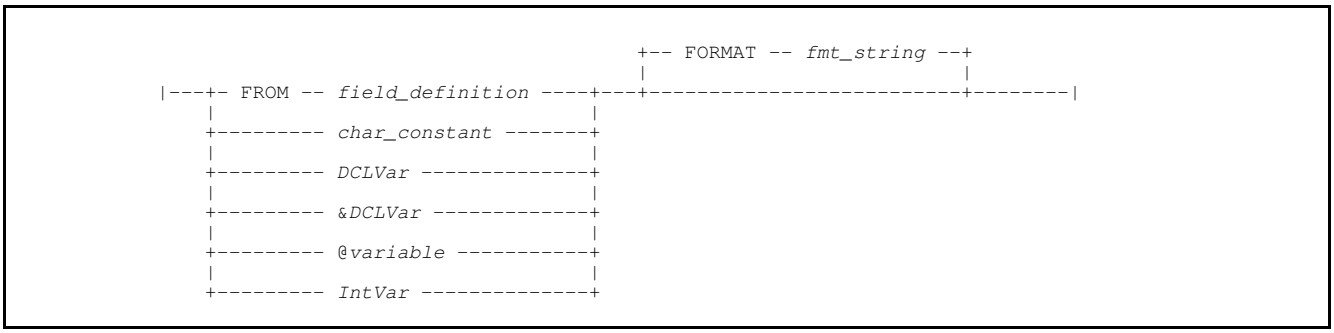
SYSTEM

Execute a command in the controlling environment in which the SELCOPY program executes.

Syntax:



Command Element:



Synonyms:

SYSTEM	SYS CMS DOS TSO UNIX

Description:

Supported in all environments except z/OS batch, the SYSTEM operation will pass a command string to the controlling environment for execution. Selection processing will pause until control is passed back to the SELCOPY program following completion of the command execution.

The system return code value set by the SYSTEM command execution is assigned to the internal variable RETSYS.

Supported environments are z/OS TSO, z/VM CMS and also Microsoft Windows and any of the Linux/Unix command shells.

Parameters:

`Command Element`
Specifies an element that constitutes a portion of the command text string.

Multiple command string elements are concatenated, in the order in which they are specified, to construct the complete command string.

Command element values that are of numeric or character numeric data type are automatically converted to decimal character display format. Use &*DCLVar* to obtain the unformatted value of a numeric *DCLVar*.

Each command element may be specified in one of the following formats:

FROM *field_definition*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type.

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL. However, if parameter FORMAT is used to convert the field data to printable hex, specification of a length is mandatory.

Although not necessary, keyword FROM may also be used before a command element specified as a *char_constant* or a *DCLVar* of character data type.

char_constant
A quoted or hexadecimal character constant.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
The name of an @variable that has a non-null value.
If *@variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
The name of a SELCOPY internal variable.

FORMAT *fmt_string*
FMAT
FMT
FORMAT specifies *fmt_string*, a format string used as the character display template for data specified by the command string element.

The source data in a command element of numeric data type is automatically converted to displayable character format (using a CVxC operation). The length of the command element is determined by *fmt_string*, not the length of the command element source.

For command elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For command elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the command element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For a command element specified as an *@variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
`'SS,SSS,SSS,SS9'`	Used for *@variable*, *IntVar* and *DCLVar* representing an integer value.
`'SS,SSS,SSS,SS9.9999'`	Used for *DCLVar* representing a rational value.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Examples:

Example 1.

The following example invokes another SELCOPY program from the current SELCOPY program in a TSO environment. Note that, the SELCOPY command line parameter "-lst" must be used to direct the list output to a different location from the current SELCOPY execution.

```
TSO     "SLC / -lst=USER123.SELC.LST2.TMP  !read indd  !print stopaft 4  !end"
```

Example 2.

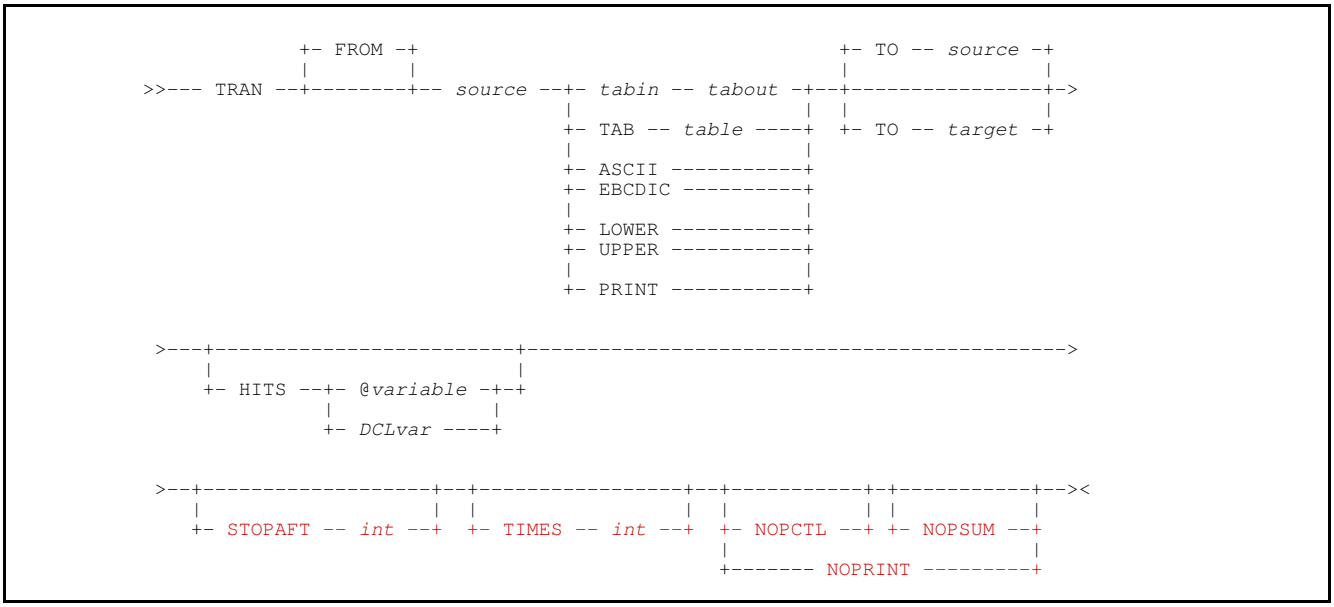
The following will issue TSO DELETE and ALLOC commands.

```
SYSTEM  "DELETE 'USER123.TEST.LOG' "                          STOPAFT 1
TSO     "ALLOC  SHR REUSE F(INDD)  DSN('USER123.TEST.RECFMV')"  STOPAFT 1
```

TRAN

Translate characters in a character source variable or field definition.

Syntax:



Synonyms:

TRAN	TR

Description:

Translate all characters belonging to a declared variable or field definition *source* value of character data type.

The hex representation (x'00' to x'FF') of each character in the *source*, identifies a code point offset into a single byte character set (SBCS) translate table of length 256. The character found at that offset in the translate table is used to replace the original character in the *source* value.

Individual character code points and their corresponding translation values may be specified using *tabin* and *tabout* parameters. All characters in the *source* value that correspond to code points not referenced by *tabin*, translate without change. i.e. SELCOPY constructs a temporary translate table whereby each code point offset references the offset value itself, then substitutes the values specified by *tabout* at corresponding offsets specified by *tabin*. e.g.

```
DECLARE  TRVAL   CHA                                    INI='AaBbCcDd'
TRAN     TRVAL    'ABC' 'XYZ'   TO   POS 11  * Position 11: "XaYbZcDd"
```

Alternatively, a 256 character translate table may be constructed as a declared variable or field definition and then referenced by TAB *table* in the TRAN operation. e.g. on an ASCII based system:

```
DECLARE  TRTAB   CHA(256)
DECLARE  TRVAL   CHA                                    INI='AaBbCcDd'

     TRTAB      =   '-'    FILL '-'          * "-" in all 256 characters.
POS &TRTAB+0x41 =   'ABC'                    * "ABC" at ASCII code points for "ABC".

TRAN     TRVAL   TAB=TRTAB      TO   POS 21  * Position 21: "A-B-C---"
```

SELCOPY also supports a number of in-built translate tables for converting between ASCII and EBCDIC encoding, upper case and lower case alpha, and for translating unprintable characters.

Parameters:

FROM *source*
References the character source value on which translation will occur. *source* may be specified as a declared variable of character data type, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *target* is implied. Otherwise, a length equal to the prevailing value assigned to variable LRECL is used.

tabin tabout
Specifies input (*tabin*) and output (*tabout*) lists of characters. Each character in *source* that matches a character specified by *tabin* is replaced by the corresponding character in *tabout*.

By default, the translate table used by the TRAN operation contains an ascending range of values from X'00' to X'FF'. Thus, every character code point in *source* would be unchanged following a translation. *tabin* specifies a series of code points within this translate table at which equivalent characters specified by *tabout* will be overlayed.

Both *tabin* and *tabout* may be specified as a character declared variable or as a quoted or hex character constant. The length of *tabin* and *tabout* must be the same otherwise ERROR 145 is returned. The first character in *tabin* gets translated to the first character in *tabout*, and likewise for second and subsequent characters in the *tabin* and *tabout* specification.

TAB *table*
Specifies *table*, the start position of a 256 character translate table to be used to translate characters in the *source* value.

table may be specified as a declared variable of character data type, or a Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition. *table* may also be specified as a Type 1 (*field_pLENn*) but without a length value.

The length of *table* must equal 256, otherwise ERROR 144 will be returned. If specified as a type 2 or 3 field definition, the length of the field must be determinable by SELCOPY at control statement analysis. i.e. an arithmetic expression (*expr*) defining the length, start and/or end position of the field must not contain variable values.)

ASCII
Indicates that SELCOPY's internal translate table for EBCDIC to ASCII conversion is to be used to translate characters in the *source* value. The SELCOPY EBCDIC to ASCII translate table is as follows:

HEX	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	CF	09	D3	7F	D4	D5	C3	0B	0C	0D	0E	0F
1-	10	11	12	13	C7	B4	08	C9	18	19	CC	CD	83	1D	D2	1F
2-	81	82	1C	84	86	0A	17	1B	89	91	92	95	A2	05	06	07
3-	E0	EE	16	E5	D0	1E	EA	04	8A	F6	C6	C2	14	15	C1	1A
4-	20	A6	E1	FE	EB	90	9F	E2	AB	8B	9B	2E	3C	28	2B	7C
5-	26	A9	AA	9C	DB	A5	99	E3	A8	9E	21	24	2A	29	3B	5E
6-	2D	2F	DF	DC	9A	DD	DE	98	9D	AC	BA	2C	25	5F	3E	3F
7-	D7	88	94	B0	B1	B2	FC	D6	FB	60	3A	23	40	27	3D	22
8-	F8	61	62	63	64	65	66	67	68	69	96	A4	F3	AF	AE	C5
9-	8C	6A	6B	6C	6D	6E	6F	70	71	72	97	87	CE	93	F1	80
A-	C8	7E	73	74	75	76	77	78	79	7A	EF	C0	DA	5B	F2	F9
B-	B5	B6	FD	B7	B8	B9	E6	BB	BC	BD	8D	D9	BF	5D	D8	C4
C-	7B	41	42	43	44	45	46	47	48	49	CB	CA	BE	E8	EC	ED
D-	7D	4A	4B	4C	4D	4E	4F	50	51	52	A1	AD	F5	F4	A3	8F
E-	5C	E7	53	54	55	56	57	58	59	5A	A0	85	8E	E9	E4	D1
F-	30	31	32	33	34	35	36	37	38	39	B3	F7	F0	FA	A7	FF

TRAN *source* ASCII is a synonym for operation CVEA *source*.

EBCDIC
Indicates that SELCOPY's internal translate table for ASCII to EBCDIC conversion is to be used to translate characters in the *source* value. The SELCOPY ASCII to EBCDIC translate table is as follows:

HEX	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	00	01	02	03	37	2D	2E	2F	16	05	25	0B	0C	0D	0E	0F
1-	10	11	12	13	3C	3D	32	26	18	19	3F	27	22	1D	35	1F
2-	40	5A	7F	7B	5B	6C	50	7D	4D	5D	5C	4E	6B	60	4B	61
3-	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	7A	5E	4C	7E	6E	6F
4-	7C	C1	C2	C3	C4	C5	C6	C7	C8	C9	D1	D2	D3	D4	D5	D6
5-	D7	D8	D9	E2	E3	E4	E5	E6	E7	E8	E9	AD	E0	BD	5F	6D
6-	79	81	82	83	84	85	86	87	88	89	91	92	93	94	95	96
7-	97	98	99	A2	A3	A4	A5	A6	A7	A8	A9	C0	4F	D0	A1	07
8-	9F	20	21	1C	23	EB	24	9B	71	28	38	49	90	BA	EC	DF
9-	45	29	2A	9D	72	2B	8A	9A	67	56	64	4A	53	68	59	46
A-	EA	DA	2C	DE	8B	55	41	FE	58	51	52	48	69	DB	8E	8D
B-	73	74	75	FA	15	B0	B1	B3	B4	B5	6A	B7	B8	B9	CC	BC
C-	AB	3E	3B	0A	BF	8F	3A	14	A0	17	CB	CA	1A	1B	9C	04
D-	34	EF	1E	06	08	09	77	70	BE	BB	AC	54	63	65	66	62
E-	30	42	47	57	EE	33	B6	E1	CD	ED	36	44	CE	CF	31	AA
F-	FC	9E	AE	8C	DD	DC	39	FB	80	AF	FD	78	76	B2	43	FF

TRAN *source* EBCDIC is a synonym for operation CVAE *source*.

LOWER
Indicates that SELCOPY's internal translate table for lower casing of alpha characters belonging to the ISO Basic Latin alphabet is to be used to translate characters in the *source* value.

The interpretation of a character code point as upper case alpha (A to Z) and the hex value of its lower case equivalent (a to z), is determined by the local code page (ASCII or EBCDIC) used by the system on which SELCOPY is running.

In all ASCII and EBCDIC code pages, the 26 upper case Latin alpha characters and their lower case equivalents are at invariant code points. In ASCII code pages, these are [x'41'-x'5A'] for upper case alpha and [x'61'-x'7A'] for lower case alpha. In EBCDIC code pages, these are [x'C1'-x'C9', x'D1'-x'D9', x'E2'-x'E9'] for upper case alpha and [x'81'-x'89', x'91'-x'99', x'A2'-x'A9'] for lower case alpha.

TRAN *source* LOWER is a synonym for operation LOWER *source*.

UPPER
Indicates that SELCOPY's internal translate table for upper casing of alpha characters belonging to the ISO Basic Latin alphabet is to be used to translate characters in the *source* value. See parameter LOWER for details of upper/lower case alpha character interpretation and code point values.

TRAN *source* UPPER is a synonym for operation UPPER *source*.

PRINT
Indicates that SELCOPY's PRINT TYPE C translate table, used to convert each unprintable character in a print element source value to "." (dot/period), is to be used to translate characters in the TRAN *source* value.

See TYPE C description of the PRINT operation for the table of characters determined as being printable for both ASCII and EBCDIC systems. This TYPE C print table, and so the TRAN PRINT operation, is also influenced by values set by the PRINTABLE and UNPRINTABLE environment options.

TO *target*
Identifies the declared variable or Type 1, 2 or 3 field definition that is the target of the operation.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *source* is implied.

If the length of *target* is less than that of *source*, then the value assigned to *target* is truncated on the right. If its length is greater than that of *source*, then the *source* length is used. In this case, text assigned to *target* that exists at locations beyond the length of *source*, remains unchanged following the TRAN operation. (i.e. No padding occurs.)

Default is TO *source*.

HITS *@variable | DCLvar*
Applicable only when *tabin tabout* is used. HITS nominates an *@variable* or declared variable (*DCLvar*) of numeric data type to be assigned a value equal to the number of characters in *source* that match a code point specified by *tabin*.

If no characters in *source* match a code point *tabin*, the value is 0 (zero).

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of TRAN for the same source and target values is unnecessary and so TIMES should never be used on the TRAN operation.
See common parameter TIMES for details.

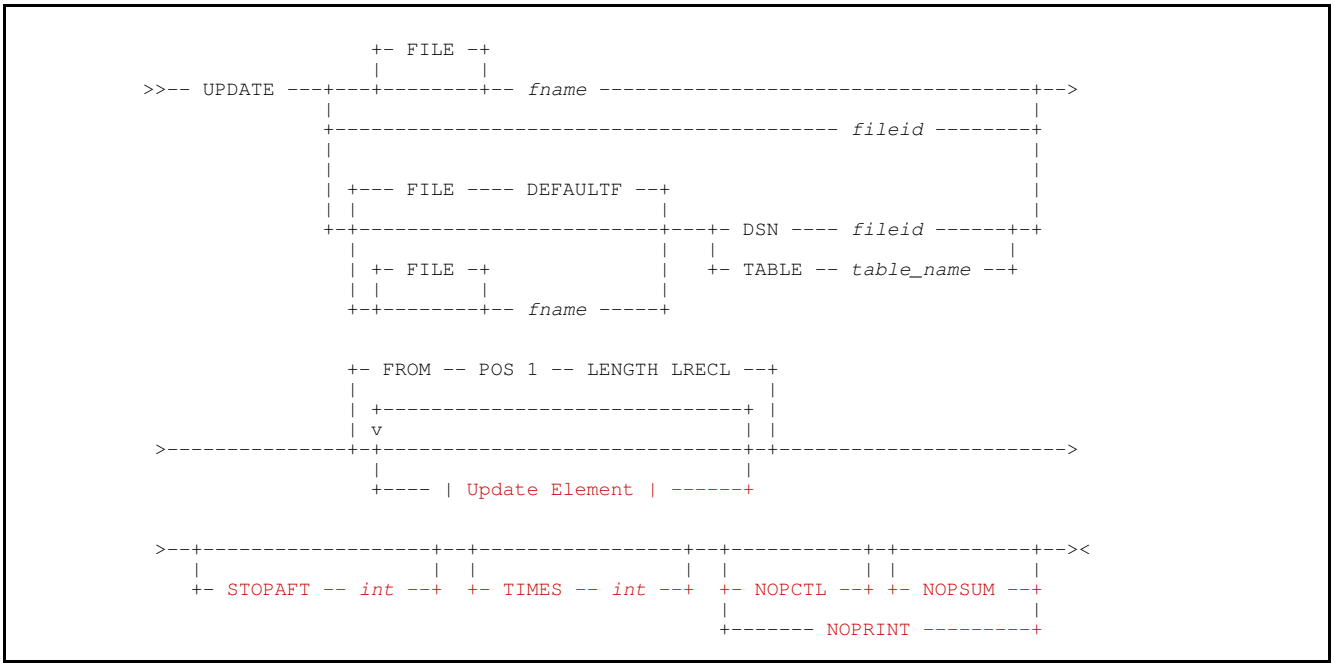
Examples:

```
DECLARE SRCE  CHAR(100)  INIT='aa bbb cccc dd eeeeee'
DECLARE STR1  CHAR       INI  'abc'
DECLARE STR2  CHAR       INI  'xyz'

TRAN    SRCE   'ad'  'ad'   HITS=@A     * Will set @A to 4, changing every
                                       * 'a' to 'a' and 'd' to 'd',
                                       * effectively changing nothing,
                                       * but you get a count of the hits.

TRAN    SRCE   STR1  STR2   HITS=@A     * Will set @A to 9 and change the
                                       * contents of SRCE:
                                       *    all 'a' chars to 'x'
                                       *    all 'b' chars to 'y' and
                                       *    all 'c' chars to 'z'

MOD     8 AT 1   'hello'  ASCII        * Position   1: "hello   " (ASCII)
TRAN    8 AT 1   UPPER    TO 261       * Position 261: "HELLO   "
TRAN    8 AT 261 EBCDIC   TO 281       * Position 281: x'C8C5D3D3,D6404040'
CVAE    8 AT 261          TO 281       * Same as above.
```

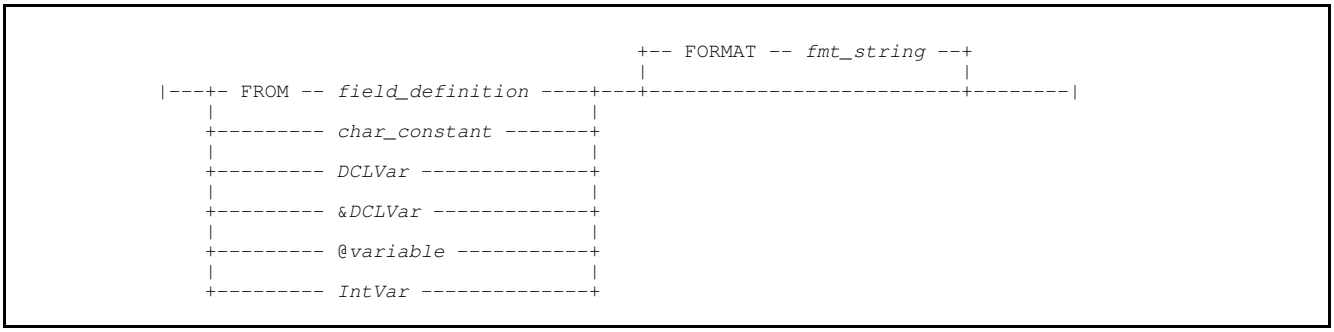
UPDATE

Update a record or table row read from an input data object source.

Syntax:



Update Element:



Synonyms:

UPDATE	UPD REP REPL REWRITE

Description:

Update of current input record is supported for z/OS data sets and library members, ODBC input tables, and for VSAM, z/VM CMS, Micro Focus, Windows and Unix-like files.

For z/OS data sets and library members, Micro Focus files and VSAM data sets, input parameter UPD must have been specified for the data object in order that it is opened for update. If not, ERROR 538 indicating failed update, or ERROR 26 indicating an input/output conflict on the same file name, will be returned. UPD may be specified on the READ or OPEN operation.

UPDATE is not supported for input read via a CAT sub-operation.

File Update

For input file objects, the UPDATE operation will replace the contents of the last record read from the specified input object with data starting at the FROM position.

For VSAM KSDS file objects only, the length of the updated record may change and so the data that replaces the original input record, has a length equal to the current value of variable LRECL. Beware that this value may be updated by another READ operation. If the LRECL of the current record exceeds the defined maximum for the KSDS data set, the record is truncated at the maximum length and return code 5 is set.

For all other file object and ODBC table object input, the length of the update data will always be the same as the length of the input record being replaced, regardless of changes to the value of LRECL.

Update of records read using the READ DIRDATA parameter is supported. However, an attempt to update a directory record will return ERROR 565.

ODBC Table Update

For ODBC database table update, the FOR UPDATE OF clause, nominating a list of updateable columns, must be specified on the SQL query that opens the table cursor. This is achieved by specifying one of the following on the READ operation:

- If parameter TABLE is used to identify a base table, specify the UPD *update_columns* parameter.
- If parameter SQL is used to specify an SQL query statement, the statement should include a FOR UPDATE OF clause. e.g.

```
DECLARE  NEWPAY   DEC(7,2)

READ     EMP      SQL='SELECT EMPNO,WORKDEPT,SALARY          \
                          FROM DSN8910.EMP               \
                          FOR UPDATE OF SALARY'
IF WORKDEPT = 'E11'
  THEN NEWPAY = SALARY
  THEN ADD 1000 TO NEWPAY                  * Give WORKDEPT 'E11' a pay rise.
  THEN CVPC NEWPAY TO SALARY FMT='99999.99' * Convert it back to character.
  THEN UPD EMP                             * Do the update.
```

As for input row values, the updated column values must be supplied in fixed length character data format and must also have the same relative positions as the input values. SELCOPY builds the SQL UPDATE statement assignment clause using these values and the updateable column names to which they belong.

UPDATE performs the positioned form of an SQL UPDATE statement on the current row and can only update columns belonging to the first (or only) source table referenced by the SQL query FROM clause. The searched form of SQL UPDATE may be actioned using the ODBC operation.

Parameters:

fileid

For file object update where no specific *fname* has been defined on the READ operation, *fileid* may be used to identify the same file referenced on READ. *fileid* is a fileid clause specifying the name by which the input file is known to the local system.

If *fname* is specified and is already associated with a *fileid*, then re-specification of *fileid* on the UPDATE operation is unnecessary.

If not specified as the DSN parameter value, then specification of FILE is invalid and the associated *fname* is derived from *fileid* as described by *fileid* for the READ operation.

fileid may be specified as an unquoted literal, a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If specified as a variable or field definition, *fielid* must be an argument to parameter DSN. Also, *fileid* specified on the READ operation for the input file, must also be a variable or field definition (i.e. a dynamic value).

DSN *fileid*

Specifies the name of the input file. See *fileid* for details.

If no *fname* is specified on an UPDATE operation which uses DSN, then an *fname* of DEFAULTF is used by default.

FILE *fname*
F

Identifies the file name assigned to the input data object to be updated.

fname must match the specified (or derived) file name on the READ operation for the object being updated. It may only be specified as an unquoted literal.

The *fname* value may be specified with or without the FILE parameter keyword.

Update Element
Specifies an update element that constitutes a portion of the update record data.

Multiple update element specifications are concatenated with no intervening blanks and in the order in which they are specified to construct the complete update record. The combined lengths of each of the elements define the update record length. Note that, for ODBC table update, the updated column data must be at the same relative offsets as the input columns.

If no update element is specified, the update record data defaults to a single field definition starting at position 1 with length equal to the prevailing value of the variable LRECL. (i.e. FROM POS 1 LENGTH LRECL).

All update element values that are of numeric or character numeric data type are automatically converted to decimal character display format before being written to the update data object (see the FORMAT parameter). Use &*DCLVar* to output the unformatted value of a numeric *DCLVar*.

Each update element may be specified in one of the following formats:

FROM *field_definition*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type.

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL. However, if parameter FORMAT is used to convert the field data to printable hex, specification of a length is mandatory.

Although not necessary, keyword FROM may also be used before an update element specified as a *char_constant* or a *DCLVar* of character data type.

char_constant
A quoted or hexadecimal character constant.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
The name of an @variable that has a non-null value.
If *@variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
The name of a SELCOPY internal variable.

FORMAT *fmt_string*
FMAT
FMT
FORMAT specifies *fmt_string*, a format string used as the character display template for data specified by the update element.

The source data in an update element of numeric data type is automatically converted to displayable character format (using a CVxC operation) before it gets written. The length of the update element is determined by *fmt_string*, not the length of the update element source.

For update elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For update elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the update element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For an update element specified as an *@variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
`'SS,SSS,SSS,SS9'`	Used for *@variable*, *IntVar* and *DCLVar* representing an integer value.
`'SS,SSS,SSS,SS9.9999'`	Used for *DCLVar* representing a rational value.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TABLE *table_name*
TAB

For ODBC table object update where no specific *fname* has been defined on the READ operation, TABLE *table_name* may be used to identify the same base table referenced by the READ TABLE parameter.

If *fname* is specified and is already associated with a *table_name*, then re-specification of *table_name* on the UPDATE operation is unnecessary.

If no *fname* is specified on an UPDATE operation which uses TABLE, then an *fname* of DEFAULTF is used by default.

The *table_name* value may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If *table_name* is specified as a variable or field definition, then *table_name* specified on the READ operation for the same input table, must also be a variable or field definition (i.e. a dynamic value).

TIMES *int*

Multiple execution of UPDATE for the same data is unnecessary and so should never be used. See common parameter TIMES for details.

Examples:

Example 1.

Update a very large CMS file so that ':' in position 1 of a record becomes '-'.

```
READ BIGFILE.DATA.D2

IF     POS 1 = ':'
  THEN POS 1 = '-'                          * Make the change.
  THEN UPDATE BIGFILE.DATA.D2               * Rewrite the record.
  THEN PRINT                                * Print all updated records.
  THEN WRITE  CHANGES.DATA.A1               * Also write a "Changes" file.
  THEN LOG   STOPAFT 4                      * Log first 4 for interest.
```

Example 2.

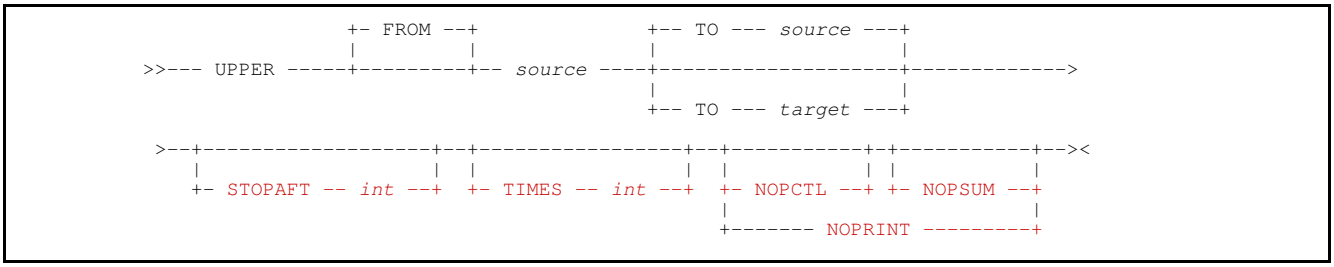
Process a VSAM KSDS data set in a z/OS TSO environment, prompting the user for each record key to be flagged. The selected record is read directly by key and the user prompted to confirm the change. If so, the record is updated in positions 27-39 with the character string "==FLAGGED==".

```
OPTION   WORKLEN 9000

READ  INKS            STOPAFT 1               * Dummy read.
LOG 'Enter key to update..'   REPLY 6 AT 2000  * Prompt user for record key.

IF POS 2000   = 'QUIT  '                      * Check for "QUIT".
  THEN GOTO EOJ                               * End the job.

READ  INKS                      KEY 6 AT 2000 * Read the requested record.
IF POS 1 = '--- KEY/REC NOT FOUND ---'        * If not found...
  THE LOG                                     * Tell the user.
  THEN GOTO GET                               * Goto back to first prompt.

LOG         LENGTH 50  TYPE=C                 * Display the record data.
LOG ' '                                       * Blank line.
LOG 'Update above? Yes/No..'  REPLY 3 AT 2010 * Verify update.
UPPER                         3 AT 2010       * Force Upper case.

IF POS 2010   = 'YES'                         * If confirmed...
  THEN POS 27 = '==FLAGGED=='                 * Insert the eye-catcher.
  THEN UPDATE INKS                            * Update the input record.
```

UPPER

Upper case alpha text in a character source value.

Syntax:



Description:

All lower case alpha characters belonging to the ISO Basic Latin alphabet that occur within text represented by the *source* character value, are upper cased and the value assigned to *target*. The *target* location may overlap storage at the *source* value location. If *target* is not specified, then the *source* also becomes the target of the operation and its value is updated.

The interpretation of a character as alpha depends on the local code page used by the system on which SELCOPY executes. In all ASCII and EBCDIC code pages, the 26 lower case Latin alpha characters (a to z) are at invariant code points. i.e. ASCII [x'61'-x'7A'] and EBCDIC [x'81'-x'89', x'91'-x'99', x'A2'-x'A9']

Each occurrence of a lower case alpha character will translate to its equivalent upper case representation (A to Z). These characters also exist at invariant code points. i.e. ASCII [x'41'-x'5A'] and EBCDIC [x'C1'-x'C9', x'D1'-x'D9', x'E2'-x'E9']

```
MOD   50 AT 1                  "The Quick Brown Fox Jumped over the lazy dog."
UPPER 50 AT 1  TO 51
PRINT FROM  50 AT 51  * Prints: "THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG."
```

The UPPER operation is a synonym for the TRAN operation with parameter UPPER.

If upper casing is required for extended Latin characters or characters belonging to other alphabets, then the TRAN operation must be used. The translate table referenced by the TRAN operation would need to identify each character's upper case code point at its lower case code point location within the table.

Parameters:

FROM *source*
References the character text value in which alpha characters will be upper cased. *source* may be specified as a declared variable of character data type, or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) character field definition.

If TO *target* is specified, *source* may also be specified as a quoted or hex character constant.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *target* is implied. Otherwise, a length equal to the prevailing value assigned to variable LRECL is used.

TO *target*
Identifies the declared variable or Type 1, 2 or 3 field definition that is the target of the operation. If *source* is a constant value, then TO *target* is mandatory.

If a field definition of type 1 (*field_pLENn*) is specified without a length, then the length of *source* is implied.

If the length of *target* is less than that of *source*, then the value assigned to *target* is truncated on the right. If its length is greater than that of *source*, then the *source* length is used. In this case, text assigned to *target* that exists at locations beyond the length of *source*, remains unchanged following the UPPER operation. (i.e. No padding occurs.)

Default is TO *source*.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

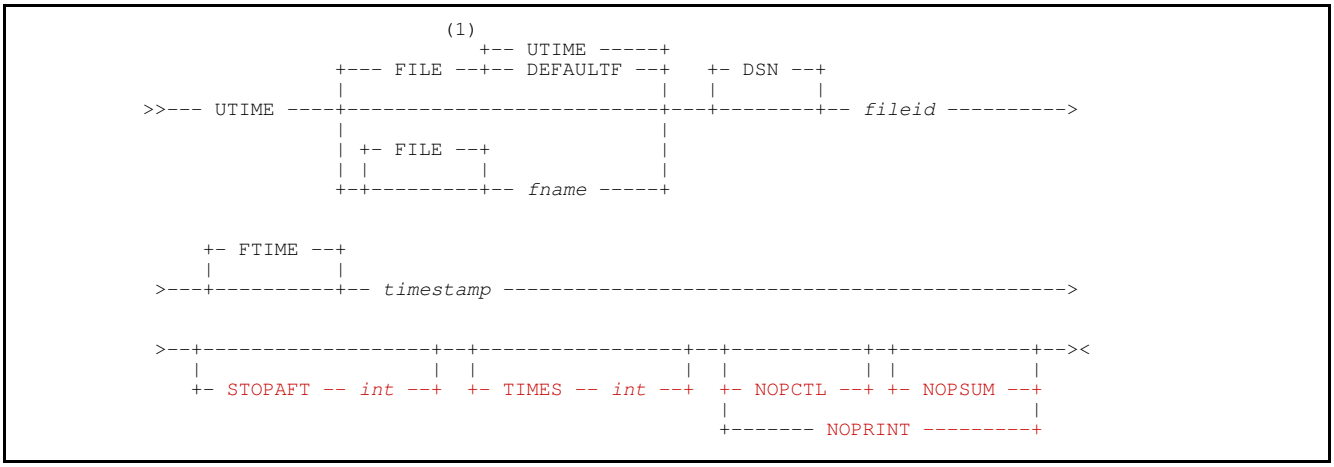
STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of UPPER for the same source and target values is unnecessary and so TIMES should never be used on the UPPER operation. See common parameter TIMES for details.

UTIME

Set the access and modification timestamp for a z/VM CMS, Windows or POSIX file.

Syntax:



Syntax Notes:

(1) The default value of *fname* is DEFAULTF if the keyword DSN is used, otherwise UTIME.

Description:

The UTIME operation is supported in z/VM CMS, Windows and POSIX environments only.

Each execution of UTIME will update the Access and Modification *timestamp* recorded for the specified *fileid*. If the timestamp update fails (e.g. if the file identified by fileid does not exist), then return code 8 is set.

Parameters:

fileid

Identifies the Windows or Unix-like file to which the new *timestamp* will be set. *fileid* is a fileid clause specifying the name by which the file is known to the local system.

If not specified as the DSN parameter value, then specification of FILE is invalid and an *fname* of **UTIME** is used by default.

fileid may be specified as an unquoted literal, a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Notes on *fileid* specification:

1. If *fileid* is specified as a declared variable or field definition, then it must be an argument to parameter DSN.
2. In a z/VM CMS environment, where *fileid* may be interpreted as either a CMS mini disk file specification or a BFS Unix-like file name, then SELCOPY will treat *fileid* first as a mini disk file and then, if not found, as a Unix-like file name.
3. A Windows or Unix-like *fileid* is not upper cased, even if specified as an unquoted literal.

DSN *fileid*

Specifies the name of the file. See *fileid* for details.

If no *fname* is specified on a UTIME operation which uses DSN, then an *fname* of DEFAULTF is used by default.

FILE *fname*
F

Identifies the name to be assigned to the file. See description of File Name for further details.

The value *fname* is a programmer defined name to be assigned to the file specified by the DSN parameter. *fname* can only be specified as an unquoted literal.

The *fname* value may be specified with or without the FILE parameter keyword.

FTIME *timestamp*
Specifies *timestamp*, an International Standard (ISO) date (ccyymmdd) and time (HH:MM:SS) to be assigned as the file's timestamp value.

timestamp may be specified as an unquoted literal, a quoted character constant, a declared variable of character data type or a Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

The *timestamp* ISO date must contain a 4-digit year but may omit a time value, in which case a default time of 00:00:00 is applied. Any of the characters "." (dot/period), ":" (colon), "/" (slash) and/or "-" (minus/hyphen) may be specified within the *timestamp* value. Delimiter characters "," (comma) and b (blank) may also be used to apply punctuation if *timestamp* is specified as any of the supported formats other than an unquoted literal. e.g.

```
  UTIME   DSN='SSUTIME2.tmp'  FTIME='2000/12/29 00.33.22'
  UTIME        SSUTIME2.tmp         '2000-12-29    00:33:22'  * Extra blanks.
  UTIME        SSUTIME2.tmp         '2000.12.29,00/33/22'
  UTIME        SSUTIME2.tmp          2000/12/29...00:33:22    * Unquoted literal.
```

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
Repeated execution of UTIME for the same timestamp and fileid is unnecessary and so TIMES should never be used on UTIME. See common parameter TIMES for details.

Example:

The following example illustrates use of UTIME to set the timestamp of a number of files whose fileids, and new timestamps, are supplied as card input.

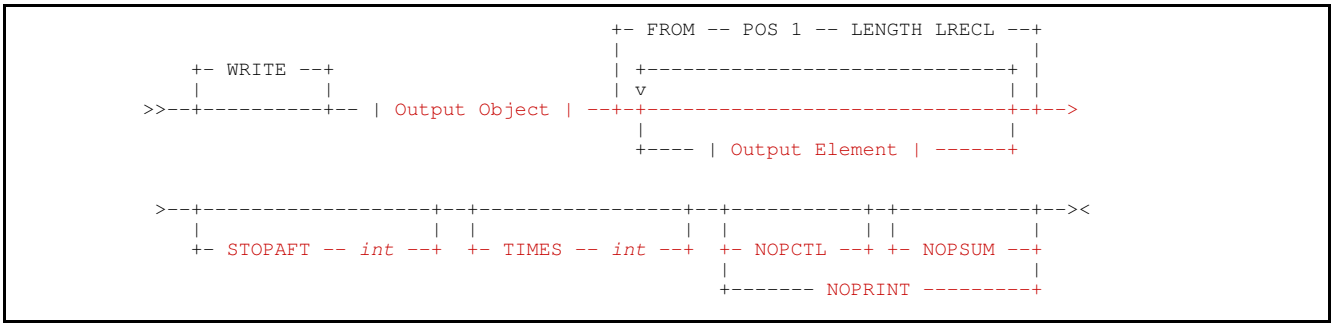
```
 read card   fill                    * Or some other file, possibly SELCOPY's DIR input.

 if pos  5    <> '/'
   then goto get                     * (Further selection or modification here perhaps.)
                                     * (      e.g. Repairing GMT TZ problems.          )

 if pos 31,80 <> ' ' reverse ptr=@e  * Find end of filename. Must omit trailing blanks.
   then utime  31,@e      1,19       * Update the timestamp for this file.

end
* ..,....1....,....2....,....3....,....4..
* Reqd Timestamp.              On file.
2000/12/29 00:33:22          g:\abc\xxxxx\some.file
2001/02/09 01.20.38          g:\abc\xxxxx\some.other.file
2001/02/09 01.31.51          g:\abc\xxxxx\file3
```

Figure 23. UTIME operation sample usage.

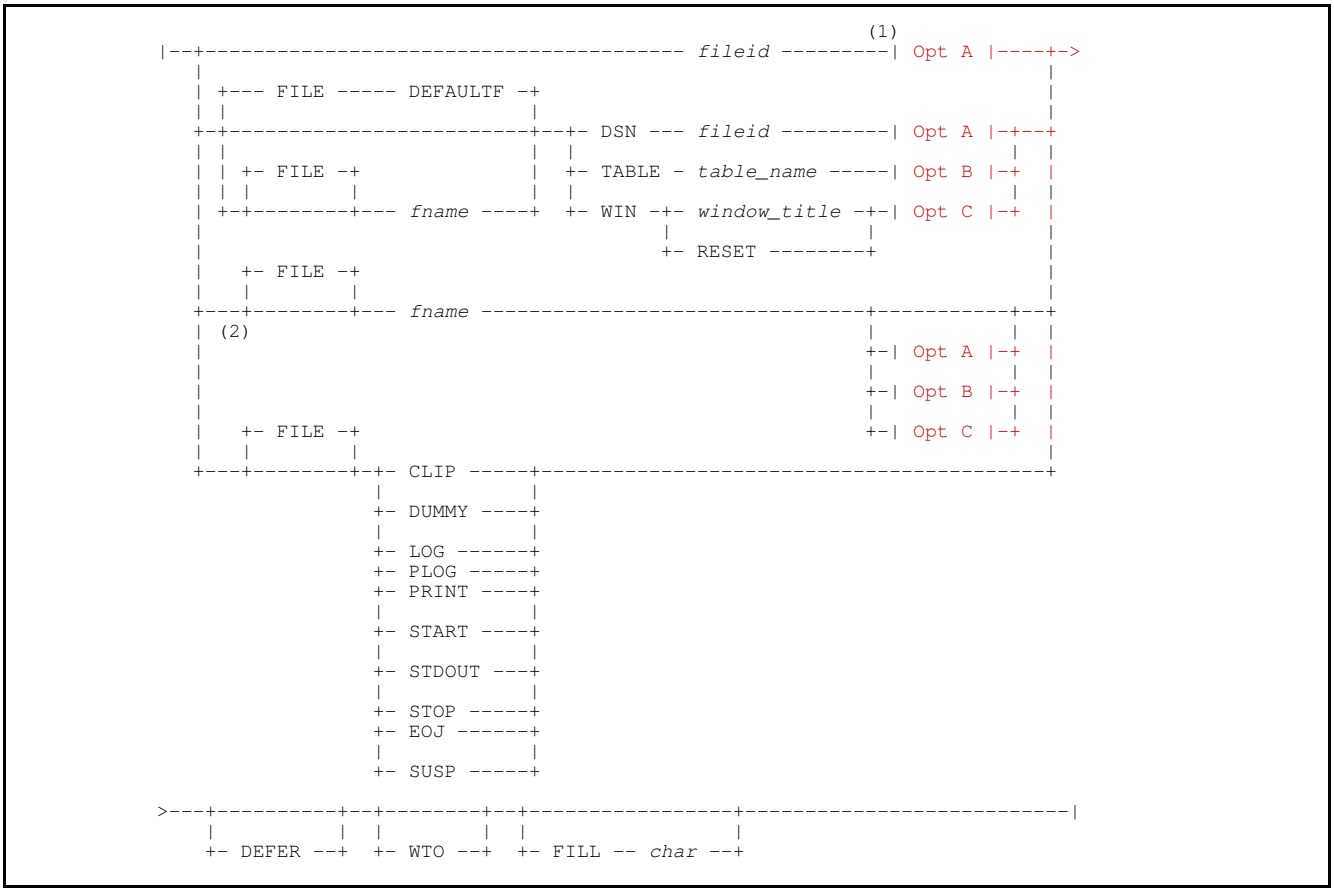
WRITE

Output data to a data object from a location in storage.

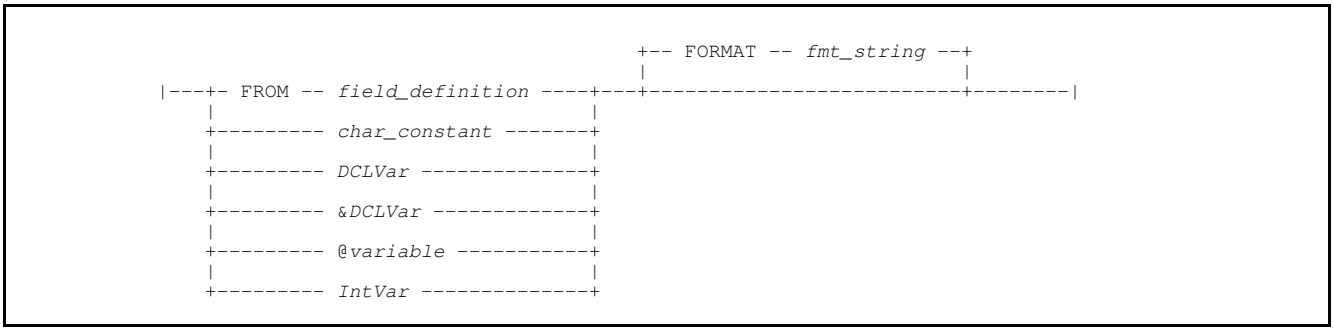
Syntax:



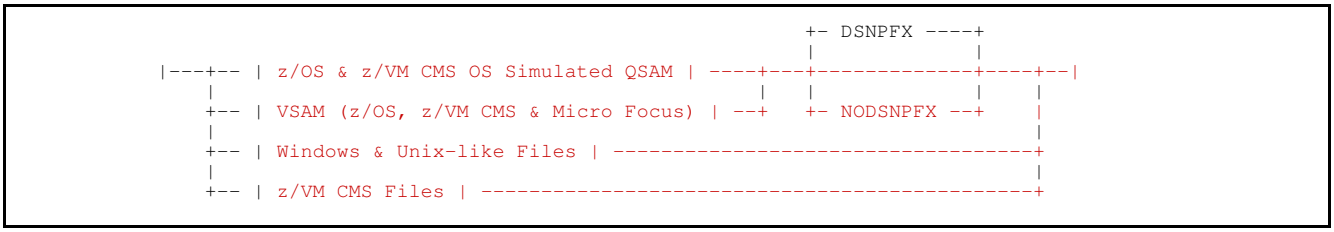
Output Object: (Dataset/File, Database Table or Focus Window)



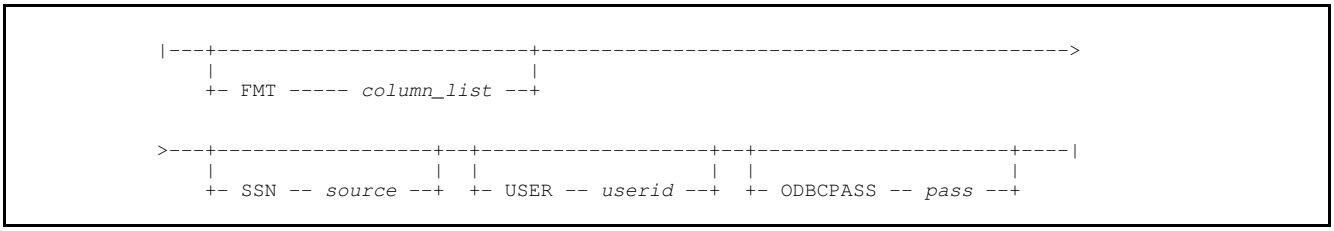
Output Element: (Output Record/Keystroke Data)



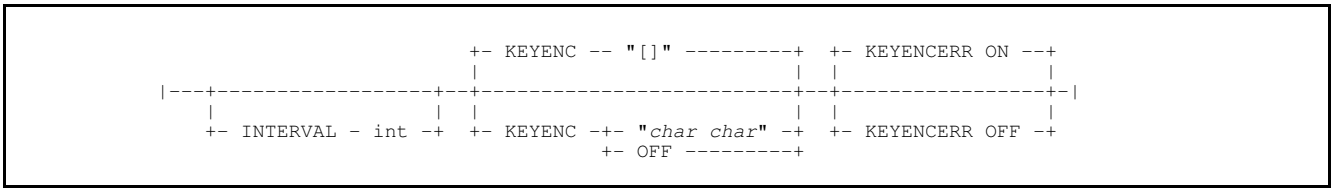
Opt A: (Dataset/File Object Options)



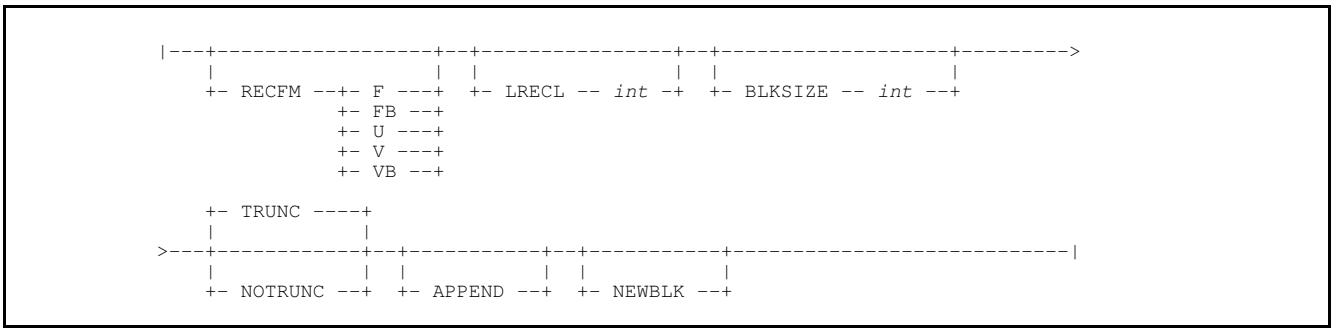
Opt B: (Database Table Object Options)



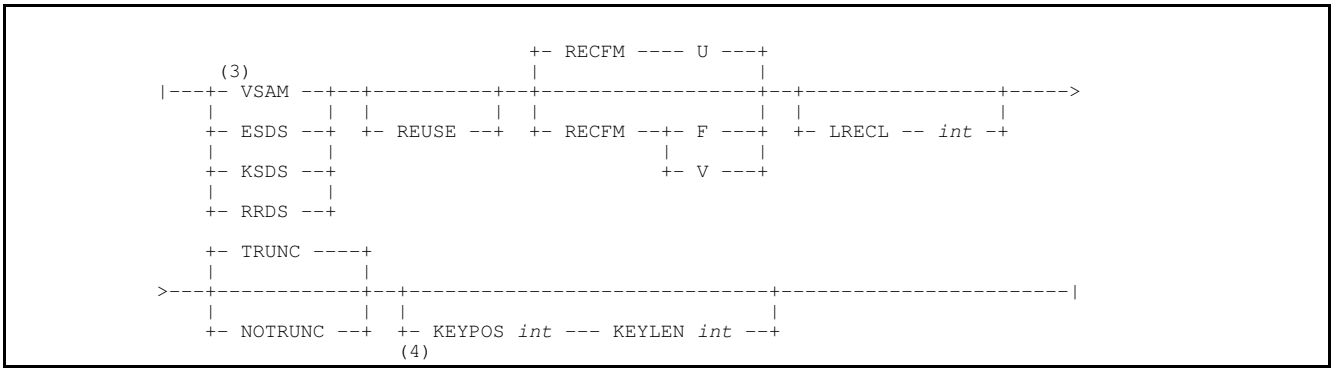
Opt C: (Window Keystroke Options)



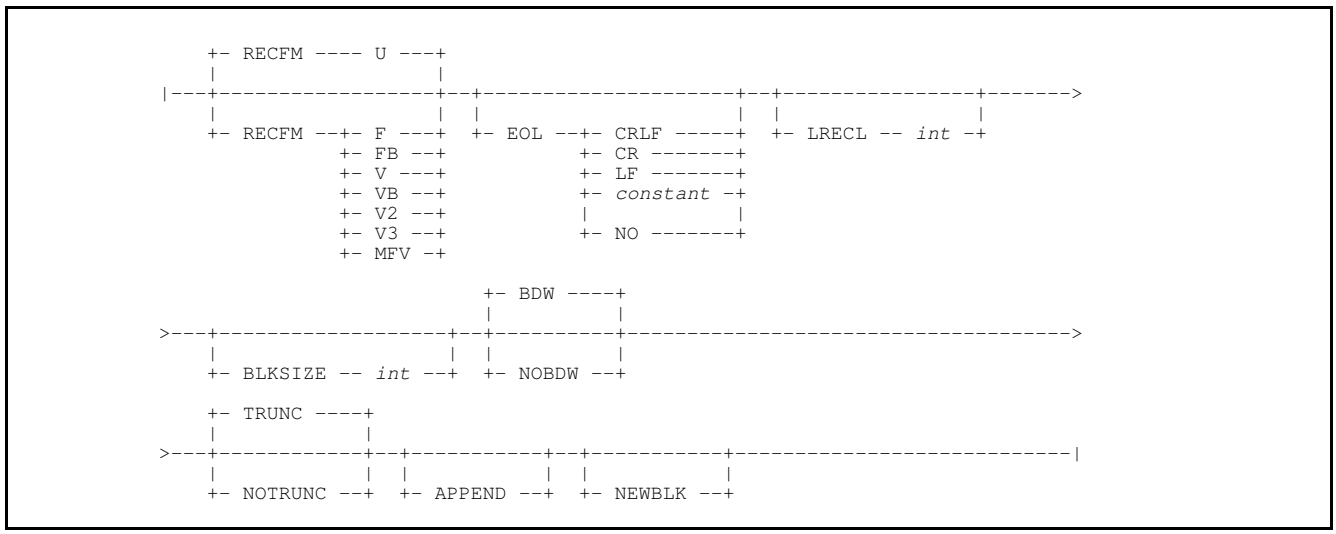
z/OS & z/VM CMS OS Simulated QSAM:



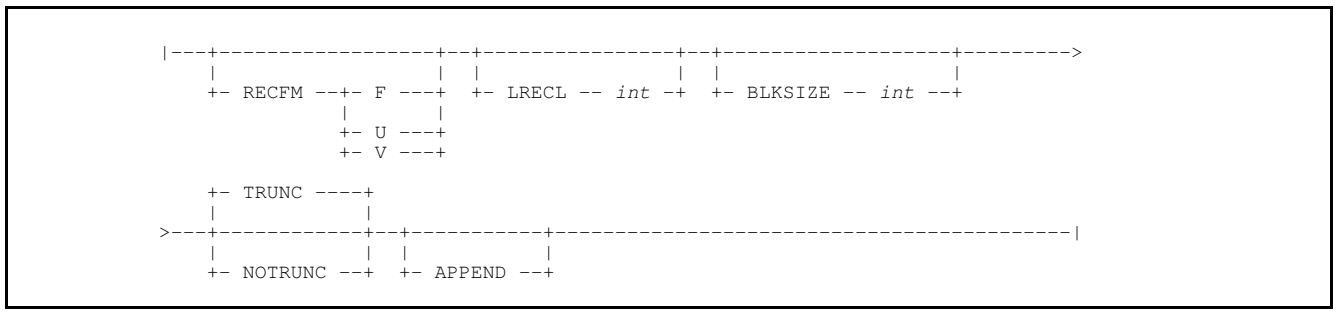
VSAM (z/OS, z/VM CMS & Micro Focus):



Windows & Unix-like Files:



z/VM CMS Files:



Syntax Notes:

(1) Optional parameters may be specified on any WRITE, OPEN or CLOSE operation for a data object referenced by the same *fname*. If different option values relating to file geometry are specified for the same *fname* (e.g. RECFM, LRECL, BLKSIZE, VSAM), then the last value specified will be used.

(2) WRITE to *fname*, without a reference to an input data object, is valid only if *fname* has already been associated with a data object. i.e.

1. *fname* is an already allocated z/OS or z/VM CMS ddname.
2. Control statement analysis processing has already occurred for an OPEN, CLOSE or WRITE operation that associates *fname* with a data object.

(3) Specification of VSAM, KSDS, ESDS or RRDS is not necessary for z/OS VSAM data sets. For z/OS, the VSAM data set type is established dynamically from the ICF catalog.

(4) KEYPOS and KEYLEN are used only on output to a new Micro Focus indexed (KSDS) file.

Synonyms:

WRITE	WR PUT (**For ODBC output only:** INSERT INS ISRT)

Description:

Each execution of a WRITE operation will write a single line of data to the output data object. The length of this data is determined by the type of output object, the lengths of the output elements that comprise the output data and the output parameter options specified on WRITE.

For z/OS and z/VM CMS file output (including output to VSAM data sets), each line of data is a record as defined by the data set or file organisation. For Windows and Unix-like files (including CLIP and STDOUT input), a record is usually a line of data terminated by end-of-line characters, though fixed and other variable length formats are also supported. The row of data written to an ODBC table is of fixed length, determined by the column widths selected.

A line of data written to a window object is not a record, but is instead interpreted by SELCOPY as individual keystrokes that get actioned at the focus window, as though typed at the local keyboard. (Note that the focus window is identified by the WIN

window_title parameter.)

With the exception of ODBC tables, VSAM and Micro Focus files, where output records are added to existing object data, records written using the WRITE operation will replace all existing records in the output data object. This is also true if parameter REUSE has been specified on output to a reusable VSAM data set. Where applicable, parameter APPEND may be used to preserve existing records and so write records following the last record of the data set.

Unless the output file object is unblocked, file output (including STDOUT) involves copying the records to an output buffer which is allocated to the output file object *fname* when it is opened. When an output record is copied that exceeds the length remaining in the output buffer, the contents of the buffer are physically written to the file object, the buffer is re-initialised and the output record inserted at the start of the buffer. Each execution of the WRITE operation to an unblocked file performs a physical write to the file with no buffering.

For files that are managed by QSAM, VSAM or CMS file I/O, buffering is handled by the access method itself. For Windows and Unix-like files, buffering is controlled by SELCOPY.

Parameters specified on WRITE apply either to the object open, data output or the execution of the individual SELCOPY operation.

Object Open Parameters

By default, an output data object on WRITE is opened at the end of SELCOPY's control statement analysis. Parameters affecting object open are applied once when the object is opened.

Open parameters, specified on WRITE operations for the same *fname* that occur in subsequent control statements, may update or expand on the values set by parameters in the first WRITE operation. Open parameters are as follow:

fileid	DSNPFX	KSDS	TABLE
DEFER	ESDS	REUSE	VSAM
DSN	FMT	RRDS	WIN

Open of a data object assigned to *fname* may be deferred until selection time processing. Furthermore, the object may be closed and re-opened if required. In these cases, SELCOPY uses the prevailing values of open parameters specified on all WRITE or OPEN operations for *fname*.

Deferred open may be achieved using any of the following methods:

- Specify DEFER on any WRITE operation for *fname*.
- Include an OPEN operation for *fname*.
- Specify a dynamic value (i.e. a declared variable name or field definition) for the output data object specification parameters: DSN *fileid*, FMT *column_list* and TABLE *table_name*.

Data Output Parameters

Data output is performed when the WRITE operation is executed during selection time processing and may involve additional processing by SELCOPY. e.g. The automatic addition of variable record format RDWs or, for Windows and Unix-like files, end-of-line characters.

Data output parameter values are established during control statement analysis. Therefore, parameter values specified on WRITE operations for the same *fname* that occur on subsequent control statements, may update or expand on values set by previous WRITE operations. The prevailing parameter values are used for all data output performed for *fname*. Data output parameters are as follow:

APPEND	FILL	ODBCPASS	TRUNC / NOTRUNC
BDW / NOBDW	KEYLEN	RDW / NORDW	USER
BLKSIZE	KEYPOS	RECFM	
EOL	LRECL	SSN	

Operation Specific Parameters

Operation specific parameters apply only to the individual WRITE operation. If specified on other WRITE operations for the same *fname*, these parameter values do not affect the execution of the current WRITE operation. Operation specific parameters are as follow:

FROM	KEYENC	NEWBLK	TIMES
INTERVAL	KEYENCERR	STOPAFT	WTO

Parameters:

fileid
Identifies the WRITE operation as being for file output. *fileid* is a fileid clause specifying the name by which the output file is known to the local system.

If *fileid* is the name of a z/OS sequential data set or the name of a PDS library and member, then the sequential or library data set must already be cataloged. In all other instances, SELCOPY will create a file named *fileid* if it does not already exist.

If *fileid* is not specified as the DSN parameter value, then specification of FILE is invalid and the associated *fname* used by SELCOPY to reference *fileid* is derived from *fileid* itself. The derived fname is determined as follows:

- ◊ For z/OS, if *fileid* is the name of a sequential data set or a PDS/PDSE library and member, *fname* is set to be the first qualifier of the data set name.
- ◊ For z/VM, if *fileid* is the name of a CMS file (*fn ft fm*) , *fname* is set to be the file name qualifier (*fn*).
- ◊ For Windows and Unix-like files, the derived *fname* depends on the length of the portion of *fileid* that follows the last "." (dot/period), if any. In Windows, this is the fileid extension.

    1. If this length is 3 characters or less, *fname* is the up-to-8 character file name string that follows the path specification (if any) up to, but not including, "." (dot/period) prefixed fileid extension.
    2. Otherwise, *fname* is the character string which occupies up-to-8 characters of *fileid*, not including the path specification (if any).

On z/OS systems, if *fileid* is the name of a sequential data set or PDS/PDSE library and member, then the derived *fname* is also the ddname that SELCOPY dynamically allocates to *fileid*. (see FILE parameter).

fileid may be specified as an unquoted literal, a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Notes on *fileid* specification:

1. If *fileid* is specified as a declared variable or field definition, then it must be an argument to parameter DSN. Also, open of the output file is deferred until the WRITE (or OPEN) operation is executed during selection time processing.
2. In z/OS, where *fileid* may be interpreted as being either a z/OS data set name or an HFS/ZFS Unix-like file name, then SELCOPY will treat *fileid* first as a data set name and then, if not found, as a Unix-like file name.

    Similarly in a z/VM CMS environment, where fileid may be interpreted as either a CMS mini disk file specification or a BFS Unix-like file name.
3. On a z/OS system, if environment option DSNPFX is in effect (the default), then the TSO prefix (TSO/E) or ACF userid (batch) will be prefixed to *fileid* if the constant, variable or field definition value used is unquoted. Furthermore, if neither DSNPFX nor NODSNPFX is specified on the WRITE operation, open of a data set without a DSN prefix will be attempted if the initial attempt to open a data set with a DSN prefix fails.

    Only if this second attempt to open a data set also fails, will *fileid* be treated as a Unix-like HFS/ZFS file name as described in note 2. above.
4. A Windows or Unix-like *fileid* is not upper cased, even if specified as an unquoted literal.

Output Element
Specifies an output element that constitutes a portion of the output record data.

Multiple output element specifications are concatenated with no intervening blanks and in the order in which they are specified to construct the complete output record. The combined lengths of each of the elements define the output record length.

If no output element is specified, the output record data defaults to a single field definition starting at position 1 with length equal to the prevailing value of the variable LRECL. (i.e. FROM POS 1 LENGTH LRECL).

All output element values that are of numeric or character numeric data type are automatically converted to decimal character display format before being written to the output data object (see the FORMAT parameter). Use &*DCLVar* to output the unformatted value of a numeric *DCLVar*.

Each output element may be specified in one of the following formats:

FROM *field_definition*
A Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition of any data type.

If specified as a type 1 field definition but without a length, the field will have a default length equal to the prevailing value of variable LRECL. However, if parameter FORMAT is used to convert the field data to printable hex, specification of a length is mandatory.

Although not necessary, keyword FROM may also be used before an output element specified as a *char_constant* or a *DCLVar* of character data type.

char_constant
A quoted or hexadecimal character constant.

DCLVar
The name of a previously defined declared variable of any data type.

&DCLVar
The & (ampersand) prefixed name of a previously defined declared variable of any data type. Returns the unformatted value of the variable as found in the variable's source field.

@variable
The name of an @variable that has a non-null value.
If *@variable* is null, return code 8 is set and the element value of "*?*" is used.

IntVar
The name of a SELCOPY internal variable.

FORMAT *fmt_string*
FMAT
FMT
FORMAT specifies *fmt_string*, a format string used as the character display template for data specified by the output element.

The source data in an output element of numeric data type is automatically converted to displayable character format (using a CVxC operation) before it gets written. The length of the output element is determined by *fmt_string*, not the length of the output element source.

For output elements of character data type, *fmt_string* must be a printable hex format string for which a CVCH (convert character to hex) operation will be performed.

For output elements of numeric or character numeric data type, *fmt_string* may be a printable hex or numeric format string. If numeric format, the output element value is converted from its source numeric data type to the decimal character display format described by the format string template.

For an output element specified as an *@variable*, *IntVar* or *DCLVar* of numeric or character numeric data type, a default *fmt_string* is used if no FORMAT parameter is specified. For *DCLVar*, this is the *fmt_string* specified on the DECLARE operation. Otherwise, the default *fmt_string* is one of the following:

fmt_string	Default Usage
`'SS,SSS,SSS,SS9'`	Used for *@variable*, *IntVar* and *DCLVar* representing an integer value.
`'SS,SSS,SSS,SS9.9999'`	Used for *DCLVar* representing a rational value.

Important note:

The location and length of the output record data, as specified by the output element may be modified by SELCOPY if the following are all true:

1. The prime input is a file object of variable record format (RECFM V, VB, V2, V3 or MFV) and environment (or READ parameter) option RDW is in effect. i.e. The 2, 3 or 4 byte record descriptor word is included as part of the input record data.
2. Only one output element is specified (or implied).
3. The output element is a field definition or declared variable of character data type.

If all the above are true, then, to account for the presence of the RDW in the input record, SELCOPY increments the address of the output record by the length of the RDW field and reduces the output record length by the same amount. e.g.

```
READ   INPV   DSN='NBJ.RECFMVB'   RDW  * LRECL = data length + 4 (RDW length).
PRINT                                  * Print record data including RDW.
WRITE  OUTF                            * Write (FROM 1+4 LENGTH=LRECL-4)
```

APPEND
APP
Supported only for output to z/OS and z/VM CMS OS simulated QSAM sequential data sets, CMS files and Windows and Unix-like files. APPEND indicates that records written by all WRITE operations for *fname* are to be appended to records that already exist in the output file object.

To support APPEND, a z/OS sequential data set must first be allocated with DISP=MOD. If APPEND is specified and SELCOPY dynamically allocates *fname* to the output data set, *fileid*, then DISP=MOD will be used instead of DISP=SHR.

BDW | **NOBDW**
Specifies whether or not the Windows or Unix-like output file of variable length record format (RECFM V or VB) is blocked.

If BDW is specified, then, when a block of records are written to the file object from the output data buffer, SELCOPY will generate a 4-byte block descriptor word (BDW) at the beginning of the block. The BDW is in the same format used by z/OS RECFM VB data sets and contains the size of the block as a number of bytes.

Default is BDW, output to a variable length record format file is blocked (RECFM VB).

BLKSIZE *int*
BLK
B
Specifies the size of the output data buffer defined for *fname*.

BLKSIZE should never be specified for output to a z/OS data set, CMS file or IBM VSAM data set. For these types of file, the output file buffer size is always determined by standard access method processing.

BLKSIZE is tolerated and ignored if specified on the WRITE operation for WIN keystroke and ODBC database table output.

For Windows and Unix-like files, including Windows clipboard (CLIP) and Micro Focus VSAM output, BLKSIZE overrides the default buffer size and, if LRECL is not specified, also imposes a maximum record length for the output record data. (see LRECL default). The size of the buffer determines the maximum size of a block of records written to a Windows or Unix-like RECFM VB file. The output buffer size is determined as follows:

1. The value specified on parameter BLKSIZE.
2. For fixed length record format (RECFM F) only, buffer size is the value specified on parameter LRECL, if greater than 2048.
3. For variable length record format (RECFM V or VB) only, buffer size is the same as that assigned to the prime input (default 2048). i.e. If BLKSIZE is specified on the prime input READ operation, the same BLKSIZE value will be used for the RECFM V or VB output.
4. 2048 bytes.

For all output record formats, if the maximum record length imposed by the BLKSIZE value is less than the value specified for LRECL, then ERROR 571 (Bufsize < LRECL+delims) is returned.

DEFER
Defer open of the data object until data is written during selection time processing.

DEFER allows the same data object to be opened for input, its data read and processed and then closed again before it is re-opened for output, all within the same SELCOPY job.

Note that DEFER is implied when an OPEN operation for the same *fname* exists within the control statements and/or *fileid* or *table_name* is dynamic (i.e. a declared variable name or field definition).

By default an output file is opened at the end of control statement analysis processing.

DSN *fileid*
Specifies the name of the output file. See *fileid* for details.

If no *fname* is specified on a WRITE operation which uses DSN, then an *fname* of DEFAULTF is used by default.

DSNPFX | **NODSNPFX**
USERID | **NOUSERID**
UID | **NOUID**
DSNPFX YES | **DSNPFX NO**
Applicable only in z/OS where *fileid* identifies an unquoted data set name belonging to a PDS/PDSE library or a sequential or VSAM data set.

If *fileid* is an unquoted literal or an unquoted value identified by a declared variable or field definition, then DSNPFX will prefix the value with the one of the following:

◊ The ACF userid assigned to the job if running in batch.
◊ The current TSO prefix if running in a TSO/E environment.

If specified on the WRITE operation, DSNPFX, NODSNPFX or one of their synonyms will override the current value of the DSNPFX environment option for that output file only. If not specified on WRITE operation and DSNPFX (YES) is the environment default, SELCOPY will first attempt to open the file using a data set name prefix. If that fails, SELCOPY will attempt to open the file without a data set name prefix.

EOL **CRLF** | **CR** | **LF** | **NO** | *constant*

Applicable only to RECFM U output of Windows and Unix-like files, EOL defines the end-of-line characters that SELCOPY will append to each output record when it is copied to the output buffer.

The representation of each EOL argument is as follows:

EOL	ASCII	EBCDIC	Description
CRLF	X'0D0A'	X'0D15'	Default for Windows, z/OS HFS/ZFS and z/VM BFS.
LF	X'0A'	X'15'	Default for Linux and Unix file systems.
CR	X'0D'	X'0D'	
constant			A quoted or hex character constant of any length.
NO			No end-of-line characters are used.

By default, SELCOPY will use end-of-line characters CRLF for Windows file output and LF for Unix-like file output.

EOL *constant* specifies a non-standard combination of characters to be used to delimit each output record. EOL NO indicates that records have no end-of-line characters and also implies parameter NOTRUNC.

ESDS

Applicable to IBM VSAM data sets (z/OS and z/VM CMS) and Micro Focus VSAM files. ESDS identifies the output file as a VSAM ESDS (entry sequenced data set) or a Micro Focus variable length record sequential file.

For existing z/OS VSAM data sets, parameter ESDS does not need to be specified as SELCOPY automatically identifies the access method and file organisation of the input file.

A new Micro Focus variable length record sequential file may be created using parameter ESDS. For all existing Micro Focus indexed, relative and record sequential files, parameters VSAM, KSDS, ESDS and RRDS are synonymous and indicate only that output is to be performed via the Micro Focus file handler.

FILE *fname* | **CLIP** | **DUMMY** | **STDOUT**
F

Identifies the name to be assigned to the output object or data stream. This name may be used to reference the output data object within other SELCOPY control statements. See description of File Name for further details.

The value *fname* is a programmer defined name to be assigned to the file, ODBC database table or window keystroke output specified by the DSN, TABLE or WIN parameter. *fname* can only be specified as an unquoted literal. Furthermore, if the operation keyword WRITE or any of its synonyms is omitted, *fname* must not match any of the following identifiers:

◊ A keyword that identifies a SELCOPY operation.
◊ An @variable, declared variable or an internal variable.
◊ A label name as referenced by a DO or GOTO operation.
◊ A substitution variable that is substituted by any invalid identifier.

For z/OS sequential data set or PDS/PDSE library member input, *fname* is the ddname that SELCOPY dynamically allocates to *fileid*. This is required since z/OS access methods use an allocated ddname to perform an open and subsequent I/O on the data set. By default the data set is allocated to *fname* with DISP=SHR. However, DISP=MOD is used if parameter APPEND is specified.

Special instances of *fname* exist to identify specific output data objects as follow:

CLIP	For Microsoft Windows only, output is to the Windows clipboard.
DUMMY	Output is to a dummy object source. WRITE DUMMY is equivalent to executing the DUMMY operation.
LOG	Output is to the standard error stream (stderr). WRITE LOG is equivalent to executing the LOG operation. In native z/OS and z/VM CMS environments, output to LOG will write to SYSOUT.
PLOG	Output is to the SELCOPY list output and the standard error stream (stderr). WRITE PLOG is equivalent to executing the PLOG operation.
PRINT	Output is to the SELCOPY list output. WRITE PRINT is equivalent to executing the PRINT operation.
START	No data is written. WRITE START is equivalent to executing the START operation.
STDOUT	Output is to the standard output stream (stdout). It is primarily used in Windows and POSIX environments to write data which is to be piped as input to another application. In native z/OS and z/VM CMS environments, output to STDOUT will write to fileid STDOUT.
STOP or **EOJ**	No data is written. WRITE STOP and WRITE EOJ is equivalent to executing the GOTO EOJ operation.
SUSP	No data is written. WRITE SUSP is equivalent to executing the SUSP operation.

The *fname* value (or any of the special instances) may be specified with or without the FILE parameter keyword.

FILL *char*
PAD

Applicable on output to fixed length record format files (e.g. VSAM RRDS and RECFM F data sets). FILL specifies *char*, a single character represented by a character constant that will be used to pad an output record to the fixed record length defined for the output file object.

Padding occurs when the length of the output data specified by the output elements, is less than the output record length.

FMT *column_list*

Applicable to ODBC table output, FMT specifies *column_list*, a number of comma separated column names belonging to the table or view specified by the TABLE parameter.

These column names identify the columns into which specific values will be inserted when the row is written to the base table(s). Columns belonging to the table or view that are not identified by *column_list* will be assigned a default value. An SQL error will occur if one or more of these columns do not support a default value.

The *column_list* value may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *column_list* is specified as a declared variable or field definition, then the object open, connection to the database and execution of the SQL INSERT is deferred until the WRITE operation is executed during selection time processing.

INTERVAL *int*
INTVL
DELAY

Applicable only to Window keystroke output in a Microsoft Windows environment, INTERVAL specifies *int*, an integer value representing the number of milliseconds (0.001 second) that must elapse before the next keystroke (or keystroke combination) is performed. i.e. the interval of time between each keystroke actioned by the output keystroke sequence specified on the current WRITE operation.

The interval value, *int*, may only be specified as a decimal integer constant.

The INTERVAL coded will take effect when the WRITE operation starts and will remain in effect for all subsequent WRITE operations to that window, unless changed by a different WRITE operation or by an occurrence of [INTERVAL=*int*] within the keystroke sequence.

Default is 10 milliseconds.

KEYENC "*char char*" | **NOKEYENC**
KEYENC OFF
KEYENC NO

Applicable only to Window keystroke output in a Microsoft Windows environment, KEYENC identifies the pair of enclosing characters required to identify the start and end of all keystroke parameters other than *key_char*. Alternatively, NOKEYENC (KEYENC OFF or KEYENC=NO) will disable use of all keystroke parameters other than *key_char*.

KEYENC and NOKEYENC will only affect the current WRITE operation and so are used to temporarily override the default set by the KEYENC environment option (default KEYENC "[]")

The KEYENC character pair "*char char*" is specified as a quoted character constant and must be one of the following:

`[]`	Square brackets		`{}`	Braces
`()`	Parentheses		`<>`	Chevrons

Any other character combination will give ERROR 167.

In the following example, "Editor" identifies the name in the Title Bar of the window to be addressed. The identification of the title is case insensitive and will still succeed if the name supplied is shorter than the actual title. The first window encountered with a title matching the title coded will be accepted and addressed.

```
WRITE  WIN="Editor"   FROM "<CR>edit x.x<CR><CR>top<CR>add 22<CR><DOWN><x=5>"   KEYENC = "<>"
```

The following demonstrates syntax executed as parameters on the SELCOPY command using the default KEYENC characters.

```
selcopy  !write win=ked  from "[cr]e x.x[cr]"   !end     * Default [] used.
```

KEYENCERR | **NOKEYENCERR**
KEYENCERR ON | **KEYENCERR OFF**
KEYENCERR YES | **KEYENCERR NO**

Applicable only to Window keystroke output in a Microsoft Windows environment.

By default, SELCOPY will treat values enclosed within KEYENC characters that do not represent a valid *key_special* or *key_combo* specification, as a sequence of *key_char* characters. In this case, the enclosing characters are treated as *key_char* characters.

KEYENCERR (KEYENCERR ON or KEYENCERR YES) indicates that return code 8 will be set for each one of these invalid specifications within the output keystroke sequence. Therefore, the number of instances is reflected in the return code count displayed against the WRITE operation in the SELCOPY list output selection summary. e.g.

```
WRITE WINP  WIN="prim"  '[CR]'   * Write to the "CMD.EXE - Primary" window.
WRITE WINP 'rem - Bad special keys: [123][XYZ] get used as data and RC=8 set.[CR]'
```

In the above example, the keystroke data written is as follows (hash "#" represents the ENTER key):

```
#rem - Bad special keys: [123][XYZ] get used as data and RC=8 set.#
```

NOKEYENCERR (KEYENCERR OFF or KEYENCERR NO) will suppress the return code 8 that would be set by this condition.

KEYENCERR and NOKEYENCERR will only affect the current WRITE operation and so are used to temporarily override the default set by the KEYENCERR environment option (default KEYENCERR ON).

KEYLEN *int*
KL

Applicable only on output to a new Micro Focus indexed file, KEYLEN specifies an integer constant value, *int*, which is the length of the key field within the record data.

See KSDS for Micro Focus indexed file output.

KEYPOS *int*
KP

Applicable only on output to a new Micro Focus indexed file, KEYPOS specifies an integer constant value, *int*, which is the position of the key field within the record data.

See KSDS for Micro Focus indexed file output.

KSDS

Applicable to IBM VSAM data sets (z/OS and z/VM CMS) and Micro Focus VSAM files. KSDS identifies the output file as a VSAM KSDS (key sequenced data set) or a Micro Focus variable length indexed file (IDXFORMAT 3, 4 or 8).

For existing z/OS VSAM data sets, parameter KSDS does not need to be specified as SELCOPY automatically identifies the access method and file organisation of the input file.

A new Micro Focus variable length indexed file may be created using parameter KSDS. To do this parameters KEYPOS and KEYLEN must also be specified to identify the position and length of the primary key within the output records. The specific format of the new file indexed file is determined by the IDXFORMAT specification in the Micro Focus file handler configuration file. This file has a fileid as assigned to environment variable EXTFH, otherwise it defaults to "extfh.cfg" .

For all existing Micro Focus indexed, relative and record sequential files, parameters VSAM, KSDS, ESDS and RRDS are synonymous and indicate only that output is to be performed via the Micro Focus file handler.

LRECL *int*
RECSIZE
L

Specifies the maximum output record length for *fname*. If specified on a WRITE operation without also specifying RECFM, LRECL implies fixed length record format (RECFM F) output.

LRECL should not be specified for output to an existing z/OS data set, z/VM CMS file or an IBM VSAM data set. For these types of file input, the maximum output record length is always determined by standard access method processing. If LRECL is specified that does not match the allocated LRECL value, then a file geometry conflict error will be returned.

LRECL is tolerated and ignored if specified on the WRITE operation for an ODBC table or an existing Micro Focus file. However, LRECL is mandatory on output to a new Micro Focus relative (RRDS) file in order to define the fixed length value.

For output to a new z/VM CMS file or output to Windows and Unix-like files, including Windows clipboard (CLIP), LRECL overrides the maximum record length imposed by the explicit or implied BLKSIZE value. The maximum output record length is determined as follows:

1. The value specified on parameter LRECL.
2. For undefined length record format (RECFM U), the maximum record data length is equal to the size of the output buffer (BLKSIZE) minus the length of the EOL character(s). If EOL is not specified for *fname*, the default EOL character length is 2.
3. For all other record formats (RECFM F, V, VB, V2, V3 and MFV), the maximum record data length is the size of the output buffer (BLKSIZE).

For fixed length record format (RECFM F) output, the maximum record length is the actual length of each output record.

For variable (RECFM V) and undefined (RECFM U) record format, an output record exceeding the maximum length will be truncated.

NEWBLK

Indicates that the a record written by this WRITE operation will occupy the first position of the output data buffer (i.e. output block).

By default, SELCOPY does not physically write the block of records stored in the output buffer to the output device until the buffer is full. i.e. when a WRITE operation is executed where the output record length exceeds the length remaining in the buffer. Following the physical write, the buffer is reset to empty so that the next output record for *fname* is written to the start of the buffer.

NEWBLK overrides this processing so that, if records exist in the output buffer, then the physical write occurs before the buffer is filled and before the new output record is written to the buffer. For RECFM VB output, a short block is written with the length of the block reflected in the generated block descriptor word (BDW).

NEWBLK is particularly useful when output is to a Windows, Unix or Linux tape device, or a z/OS or z/VM CMS ddname that has been assigned to a tape unit.

A tape device driver will automatically insert an inter-record gap (IRG) following each block of data physically written to the tape device. Therefore, NEWBLK may be used to control the size of the data block and the occurrence of an IRG.

The following Linux system example demonstrates output to a non-rewinding tape device and execution of the mt shell command from a SELCOPY program. Note that a non-rewinding tape device driver will not rewind the tape when the output is closed. In the example, a short block is written if the next output record needs to be the first record of a block.

```
READ           INX    DSN="/home/nbj/testfile"

IF   4 AT 1 TYPE=B  = -1         * x'FFFF,FFFF' => Tape Mark required.
  THEN  SYSTEM      "mt -f /dev/nst0  eof 1"
  THEN  GOTO GET

IF   4 AT 1 TYPE=B  =  1         * Record sequence within a block.
  THEN  WRITE  OTAPE  DSN="/dev/nst0"   NEWBLK   BLKSIZE=65536
  ELSE  WRITE  OTAPE
```

Also see the FLUSH operation which performs the same function as NEWBLK except that no record is written to the buffer following the physical write.

NOPCTL, NOPSUM, NOPRINT

See common parameters NOPCTL, NOPSUM, NOPRINT for details.

ODBCPASS *pass*
OPASS
PASSWORD
PASSWD
PASS

Applicable to ODBC table output, ODBCPASS specifies the password (*pass*) to be used, together with the associated user name (see USER parameter), when connecting to the data source via ODBC. Note that a user name and password is mandatory for connection to any data source.

pass must be specified as a character constant and its specification will override the prevailing value set by the ODBCPASS environment option.

RECFM **F** | **FB** | **U** | **V** | **VB** | **V2** | **V3** | **MFV**

Specifies the output record format for *fname* as follows:

F	Fixed length records.
FB	Fixed length blocked records.
U	Undefined length records.
V	Variable length records.
VB	Variable length blocked records.
V2	Variable length records with 2-byte length fields.
V3	Variable length File Transfer Protocol (FTP) Block Mode records.
MFV	Variable length Micro Focus record sequential records.

RECFM should not be specified for output to an existing z/OS data set, z/VM CMS file or an IBM VSAM data set. For these types of file input, the record format is always determined by standard access method processing (e.g. VSAM data sets are always RECFM U). If RECFM is specified that does not match the allocated RECFM value, then a file geometry conflict error will be returned.

RECFM is tolerated and ignored if specified on the WRITE operation for ODBC table or Micro Focus output. ODBC tables are always RECFM F and files processed using the Micro Focus file handler are always RECFM U.

For output to a new z/VM CMS file or output to Windows and Unix-like files, including Windows clipboard (CLIP), RECFM specifies the format of the output data.

z/VM CMS files of variable record format (RECFM V) do not contain records that include an RDW prefix. For these files, RECFM V indicates only that records need not be of the same length. Therefore, RECFM V and RECFM U are treated as being equivalent on a WRITE operation to a new z/VM CMS output file. Specification of RECFM FB and RECFM VB (blocked output) on the WRITE operation indicates the z/VM CMS OS simulated QSAM is to be used.

For variable record format to Windows and Unix-like files, SELCOPY generates an RDW in the format appropriate to the particular RECFM specification, and adds it as a prefix to the record data written to the output buffer. Furthermore, for RECFM MFV, SELCOPY generates a 128-byte header as the first output record. For undefined (RECFM U) record format, SELCOPY adds end-of-line character(s) to the end of each record, as defined by the EOL parameter, before writing it to the output buffer. Fixed (RECFM F) and fixed blocked (RECFM FB) record format are equivalent since output is always buffered. SELCOPY writes fixed record format data to the output buffer without the need for additional processing.

If output is to a new z/VM CMS file or to a Windows or Unix-like file, the default RECFM value is the same as that of the prime input file. However, if LRECL is specified, then RECFM F is implied.

See Data Record Format for details of the different RECFM types.

RRDS

Applicable to IBM VSAM data sets (z/OS and z/VM CMS) and Micro Focus VSAM files. RRDS identifies the output file as a VSAM RRDS (relative record data set) or a Micro Focus variable length record sequential file.

For existing z/OS VSAM data sets, parameter RRDS does not need to be specified as SELCOPY automatically identifies the access method and file organisation of the input file.

A new Micro Focus record sequential file may be created using parameter RRDS. Although the new file will be defined as being variable length, the output records must all be of fixed length to match those of a real VSAM RRDS data set. Therefore, parameter LRECL must also be specified to define the (maximum) record length for the file. All output records will be padded or truncated to this length.

For all existing Micro Focus indexed, relative and record sequential files, parameters VSAM, KSDS, ESDS and RRDS are synonymous and indicate only that output is to be performed via the Micro Focus file handler.

SSN *source*

Applicable to ODBC table output, SSN specifies the ODBC data source name (*source*) to which the ODBC connection will be made. Note that specification of an ODBC source name that has been defined to the local system, is mandatory for ODBC table input.

source must be specified as a character constant and its specification will override the prevailing value set by the SSN environment option.

STOPAFT *int*

See common parameter STOPAFT for details.

TABLE *table_name*
TAB

Identifies the WRITE operation as being for ODBC table output.

For ODBC database output, **TABLE** specifies the name of the table or view (*table_name*) to which rows will be inserted. SELCOPY will construct an SQL INSERT statement using the value specified on TABLE optional parameter FMT.

If no *fname* is specified for TABLE output, then an *fname* of DEFAULTF is used by default.

The *table_name* value may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

Note that, if *table_name* is specified as a declared variable or field definition, then the object open, connection to the database and execution of the SQL INSERT is deferred until the WRITE operation is executed during selection time processing. Therefore, the declared variable names, generated for each output column when the table is opened, may only be referenced in SELCOPY control statements if *table_name* is a quoted character string.

TIMES *int*

See common parameter TIMES for details.

TRUNC | **NOTRUNC**

TRUNC and NOTRUNC controls whether or not trailing blanks that exist in the output record data for the current WRITE operation, are truncated when written to a file object that supports records of different lengths.

If output is to a file object containing fixed length records or more than one output element is specified on the WRITE operation, then parameter TRUNC is ignored. Trailing blanks will be preserved in all of the output element specifications.

Unless EOL=NO is specified on output, the default is TRUNC (truncate trailing blanks).

USER *userid*

Applicable to ODBC table output, USER specifies the user name (*userid*) to be used, together with the associated password (see ODBCPASS parameter), when connecting to the data source via ODBC. Note that a user name and password is mandatory for connection to any data source.

userid must be specified as a character constant and its specification will override the prevailing value set by the USER environment option.

VSAM

Applicable to IBM VSAM data sets (z/OS and z/VM CMS) and Micro Focus VSAM files. VSAM identifies the output file as one of the supported VSAM data set types (KSDS, ESDS or RRDS) or as one of the supported Micro Focus file types (indexed, record sequential or relative).

For existing z/OS VSAM data sets, parameter VSAM does not need to be specified as SELCOPY automatically identifies the access method and file organisation of the input file.

A new Micro Focus file may be created using parameter VSAM. If parameters KEYPOS and KEYLEN are specified on any WRITE operation for *fname*, a variable length indexed file will be created. Otherwise, a variable length record sequential file is created by default.

For all existing Micro Focus indexed, relative and record sequential files, parameters VSAM, KSDS, ESDS and RRDS are synonymous and indicate only that output is to be performed via the Micro Focus file handler.

WIN *window_title* | **RESET**

Identifies the WRITE operation as being for Microsoft Windows Keystroke output.

For keystroke output, **WIN** specifies *window_title*, the title of the target window (as displayed in the window's title bar) on which focus will be placed before actioning the keystroke sequence. Alternatively, RESET returns focus to the window on which focus was placed before execution of the SELCOPY program.

The *window_title* is case insensitive and may be a substring of a window's title. If the target window of the WRITE operation is the window from which the SELCOPY program is started, then *window_title* must reference text at offset 0 (zero) from the start of the title string. Otherwise, *window_title* may reference text at any offset within the required window's title.

If more than one window title contains the substring *window_title*, then focus will be placed on the window where *window_title* occurs at the lowest offset within the title string. Furthermore, if this lowest offset is the same for the title strings of more than one window, then focus will be placed on the first of these windows for which the match was found.

If no *fname* is specified for WIN output, then an *fname* of DEFAULTF is used by default.

The *window_title* value may be specified as a quoted character constant, a declared variable of character data type or a Type 1 (*field_pLENn*), Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

ERROR 617, indicating that the window title was not found, will be returned if *window_title* is not found anywhere within the title string of any of the open windows.

WTO

Indicates that, at SELCOPY end-of-job, a summary of the number of records written to *fname* will be logged to stderr/SYSOUT. The logged output will have the following format:

```
SELCOPY/xxx r.el       #records=RECTOT  FILE=fname          yyyy/mm/dd hh:MM
```

VSAM Write

Sequential output is supported to VSAM cluster data sets that have already been defined using IDCAMS.

By default, the WRITE operation will output new records to a VSAM data set without replacing existing records. However, if the data set is defined as being reusable and parameter REUSE is specified on the OPEN or WRITE operation for *fname*, then the data set will be reset so that all existing records are lost (i.e. the high-used RBA is reset to 0 during OPEN). Writing to an empty data set or one which has been reset is equivalent to loading the file.

Output to a VSAM data set is keyed-sequential for KSDS and RRDS, and addressed-sequential for ESDS.

KSDS Write

Provided the records are written in ascending order of key sequence, records will be inserted at their appropriate location within the keyed data set.

Records may be of variable length as determined by the output *DCLvar* variables, constants and/or field definitions specified on the WRITE operation. If an output record length exceeds the maximum defined for the data set, then the record is automatically truncated at the maximum length and SELCOPY return code 5 is set.

```
SELCOPY/MVS 3.30 at CBL - Bridgend UK (Internal Only)              2015/10/29 16:11   PAGE   1
-----------------------------------------------------              ----------------   --------


           option  datawidth=65   pagedepth=9000

     1.    read         inpks  dsn='NBJ.SSVSAM.KSDS'            * First read our KSDS sequentially.

           if eof       inpks                                   * If end-of-file flagged.
     2.      then close inpks                                   * Close the input KSDS.
     3.      then space 1                                       * Print a blank line.
     4.      then do    write_routine                           * Write new records to our KSDS.
     5.      then goto eoj                                      * Must force end-of-job.

     6.    print                                                * Print existing records.
     7.    goto get                                             * Loop back to selection 1. (READ)

         ==write_routine==
           -------------
     8.    read         card                                    * Input card records.

           if eof       card                                    * If end-of-file flagged for CARD.
     9.      then goto  write_routine_end                       * Drop out of the sub-routine.

    10.    print                                                * Print the card records.
    11.    write        outks  dsn='NBJ.SSVSAM.KSDS'    defer   * Open for output and write to our KSDS.
    12.    goto         write_routine                           * Loop back to READ CARD.

          =write_routine_end=
           ------------------
    13.    return                                               * Return to main processing loop.

         end


    INPUT   SEL SEL                                                                    RECORD
    RECNO   TOT ID.          1         2         3         4         5         6       LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,  ------
        1     1   6 0000000 CBLM05 2015/10/08 CBL.INST.CBL15112.SZZSDBRM                   52
        2     2   6 0000010 CBLM08 2015/10/26 CBL.INST.CBL15112.SZZSDIST.CMX               56
        3     3   6 0000020 CBLM05 2015/10/26 CBL.INST.CBL15112.SZZSDIST.IPOPROC           60
        4     4   6 0000030 CBLM08 2015/10/19 CBL.INST.CBL15112.SZZSEXEC                   52
        5     5   6 0000040 CBLM05 2015/10/27 CBL.INST.CBL15112.SZZSLOAD                   52
        6     6   6 0000050 CBLM14 2015/10/08 CBL.INST.CBL15112.SZZSMAC                    51
        7     7   6 0000060 CBLM05 2015/10/20 CBL.INST.CBL15112.SZZSSAM1                   52
        8     8   6 0000070 CBLM14 2015/10/26 CBL.INST.CBL15112.SZZSTENG                   52

        1     1  10 0000005 CBLM05 2015/10/26 CBL.INST.CBL15112.SZZSDIST.CBLE             57
        2     2  10 0000015 CBLM14 2015/10/26 CBL.INST.CBL15112.SZZSDIST.IPO              56
        3     3  10 0000025 CBLM08 2015/10/26 CBL.INST.CBL15112.SZZSDIST.TLIB             57
        4     4  10 0000035 CBLM08 2015/10/20 CBL.INST.CBL15112.SZZSHELP.HTML             57
        5     5  10 0000045 CBLM06 2015/07/21 CBL.INST.CBL15112.SZZSLPA                   51
        6     6  10 0000055 CBLM06 2015/10/26 CBL.INST.CBL15112.SZZSPENG                  52
        7     7  10 0000065 CBLM08 2015/07/21 CBL.INST.CBL15112.SZZSSAM2                  52
                    ....,....1....,....2....,....3....,....4....,....5....,....6....,

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---
    1             8 READ INPKS       256    60 U              8 ***** NBJ.SSVSAM.KSDS
    2             1 CLOS INPKS       256    60 U              8 ***** NBJ.SSVSAM.KSDS
    3----5        1
    6----7        8

                    =write_routine=
    8             7 READ CARD      27998    57 VB             7       SYS15302.T161135.RA000.NBJ.R0118091
    9             1
   10             7
   11             7 WR   OUTKS       256    57 U              8 ***** NBJ.SSVSAM.KSDS
   12             7

                    =write_routine_end=
   13             1 =ret=


    ** SELCOPY/MVS 3.30.002  2015/09/17  Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                               ** Expiry: 14 Jul 2016 **
```

Figure 24. VSAM KSDS Table Output.

Using WRITE to output records to a KSDS data set is significantly more efficient than opening the KSDS for direct output (OPEN or READ with parameter UPD) and then executing INSERT operations. The INSERT method of adding records should only be used if either of the following is true:

1. Direct processing is required in order to issue DELETE and UPDATE operations.
2. It is not possible to sort the output records into key sequence before they are written to the KSDS data set.

If output records are not in key sequence, the WRITE operation will fail and ERROR 538 is returned.

RRDS Write

Fixed length records are written to RRDS record slots starting at the first slot of the data set. If this slot is occupied, a duplicate error occurs and the WRITE operation fails with ERROR 538.

The output records are of fixed length equal to that defined for the RRDS data set. Any output record length implied by the current value of variable LRECL, or specified by a the output *DCLvar* or field definition, is ignored.

ESDS Write

Variable length records, determined by the length of the output *DCLvar* variables, constants and/or field definitions specified on the WRITE operation, are written following the last record of an ESDS data set.

If an output record length exceeds the maximum defined for the data set, then the record is automatically truncated at the maximum length and SELCOPY return code 5 is set.

ODBC Table Write

This feature is currently supported on Microsoft Windows platforms only.

Database table output is supported for a variety of proprietary database products using ODBC (Open Database Connectivity). Using the ODBC interface, SELCOPY allows rows to be written to a database table or view.

The SELCOPY WRITE operation generates and executes a dynamic SQL INSERT statement using the specified table or view name, identified by parameter TABLE *table_name*, and a list of columns identified by parameter FMT *column_list*. Note that if FMT *column_list* is not specified, then *column_list* represents every column in the table or view, in the order defined by the table or view. The format of the SQL INSERT statement is:

```
INSERT INTO table_name (column_list) VALUES(?, ... ?)
```

Specification of a system name, ODBC userid and password is mandatory to identify the ODBC Data Source Name for the required database and to establish the user's access authority to the database tables. Similarly, a user work area must also be defined which has a length which is greater than or equal to the output record (table row) length. This length is equal to the sum of the selected column data area lengths, plus 1 for each character that separates the column data areas. If no work area is defined, ERROR 586 is returned.

The names and sequence order of columns selected for insert are identified by the FMT parameter. The column sequence defines the order in which the column data areas must exist in storage and so the order in which input values are to be specified. Columns in the table or view that are not selected by FMT, will be assigned default values (if supported by the column definition) when the row is inserted. If FMT *column_list* is not specified, then a column data area must exist for every column in the table or view, in the order defined by the table or view.

The length of an individual column's data area depends on the source data type of the column. For table columns of fixed or variable character data type, the length of the column data area is the defined (maximum) column length. For numeric data types, the length of a column data area is the minimum length required to contain the maximum and minimum values supported by the column data type, following its conversion to character numeric format. See ODBC table read for the column data area lengths required for each numeric and date data type.

The insert values must be moved to the relevant column data areas prior to executing the WRITE operation and the WRITE parameter FROM should be used to identify the location of the first column data area (default position 1).

The format of an insert value within a column data area is as follows:

1. Regardless of the data type of the column into which the value will be inserted, all values must be supplied in fixed length character format. Numeric and date values must be in character decimal format as described for ODBC table read.
2. For columns of character data type, values should be left adjusted within the column data area if leading blanks are not required.
3. Each value must be padded with blanks to the length of the column data area.

Specification of insert values is simplified by assigning the required values to the declared variables that are automatically generated by SELCOPY when the output table is opened during control statement analysis. These declared variables are of fixed character data type and have names that match the names of the columns selected for insert. See description of Column Declared Variables under ODBC table read.

```
SELCOPY/WNT 3.30 at CBL - Bridgend UK (Internal Only)              2015/10/23 18:26   PAGE   1
-----------------------------------------------------              ----------------   --------

         equ  wid      70     * Print length.
         equ  irec      1     * DBASE Input Area.
         equ  crec    201     * CARD  Input Area.
         equ  orec    401     *       Output Area for INSERT statement.

         opt  worka  2200   dw=wid  pw=94 pd=9999   noptot  logsql='c:\tmp\ssdb2m13.log'
         opt  user=XXX  odbcpass="XXX"  ssn=XXXXXXX   * UserId, Passwd, ODBC Sub-System Name.

         * Drop and recreate the tabe to ensure empty.

      1.    odbc     "drop    table SELCOPY_TEST"
      2.    odbc     "create  table SELCOPY_TEST    \
                       ( color      varchar(10),   \
                         date_sold  date,          \
                         price      decimal(15,2), \
                         notes      varchar(20) )"
      3.    odbc     "commit"


         ==populate==   * Loop to insert rows in table.
           --------
      4.    read   card     into crec   fill


           if eof card
      5.     then close fnam3
      6.     then odbc     "commit"
      7.     then space
      8.     then goto readv


      9.    do build_orec

     10.    insert fnam3  table='SELCOPY_TEST'   from orec   * No FMT, will default to all columns.
     11.    goto populate


         ==readv==      * Loop to verify all rows inserted.
           -----
     12.    read   fnamv  table='SELCOPY_TEST'   into irec    HDR    defer


           if eof fnamv
     13.     then eoj


     14.    print  length=wid  from irec    stopaft=222
     15.    goto readv

     16.  goto eoj


         ==build_orec==              ** Sub-Routine **
           ----------
               * Needed as a subrtn so that references to COLOR, DATE_SOLD, PRICE and NOTES occur
               * after the reference to INSERT fnam3 which defines the column positions.
     17.    NOTES+orec-1      = 20 at crec+60       * Len=20
     18.    PRICE+orec-1      = 20 at crec+40       * Len=17
     19.    DATE_SOLD+orec-1  = 20 at crec+20       * Len=19
     20.    COLOR+orec-1      = 20 at crec+00       * Len=10
     21.    print         length=wid  from orec    stopaft=222
     22.  =return=

         end


    INPUT   SEL SEL                                                                    RECORD
    RECNO   TOT ID.          1         2         3         4         5         6         7  LENGTH
    -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
        1     1  21 Crimson    2009-08-07   5.55            Must be ISO date fmt              80
        2     2  21 Grey       2008-12-31 395.22            Only 1 sale                       71
        3     3  21 Violet     2008-09-12  14.99            Sweet                             65
        4     4  21 Rose       2008-08-29   3.62            Gentle                            66

        1     1  14 COLOR     |DATE_SOLD |PRICE (17)       |NOTES (20)          |              61
        2     2  14 ----------|----------|-----------------|--------------------|              61
        3     3  14 Crimson   |2009-08-07|5.55             |Must be ISO date fmt|              61
        4     4  14 Grey      |2008-12-31|395.22           |Only 1 sale         |              61
        5     5  14 Violet    |2008-09-12|14.99            |Sweet               |              61
        6     6  14 Rose      |2008-08-29|3.62             |Gentle              |              61
                    ....,....1....,....2....,....3....,....4....,....5....,....6....,....7

    ** SELCOPY/WNT 3.30.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                ** Expiry: 14 Jul 2016 **
```

Figure 25. ODBC Table Output.

Window Keystroke Write

This feature is supported on Microsoft Windows platforms only.

SELCOPY is able to write keystrokes to any open window in the Microsoft Windows system in which it executes. (e.g. a web browser, email client, text editor or 3270 emulator.) The effect is identical to a user switching focus to the required application window then performing keyboard keystrokes to navigate the window display or enter text.

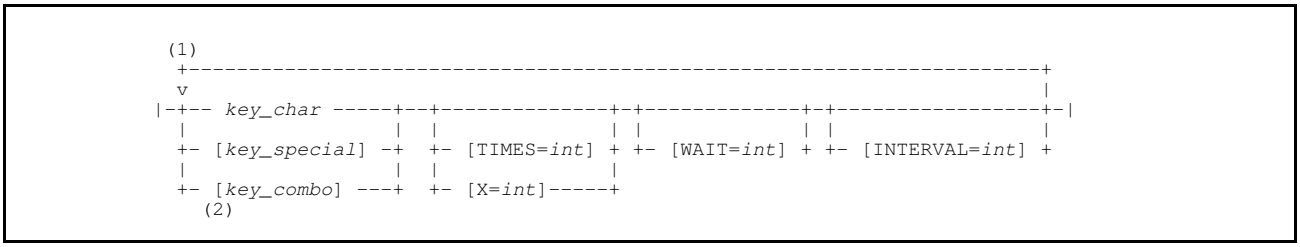
The target window is made the focus and is identified by its title (i.e. the text displayed in the title bar of a window). If not already running, an application window may be started using the SYSTEM operation.

Parameter WIN indicates that window keystroke processing is to be performed and specifies a text substring that uniquely identifies the title of the window to which keystrokes will be sent.

The output data references a string of characters which are interpreted as a sequence of keystrokes. When the WRITE operation is executed, SELCOPY executes each individual keystroke in the sequence order in which they occur in the character string.

Keystroke Syntax

The syntax of a keystroke character string is as follows:



(1) Blank characters are treated as *key_char* specifications and so cannot be used to separate keystroke parameters. However, blank separators are supported when specified within keystroke delimiter characters (as defined by KEYENC).

(2) WRITE parameter KEYENC identifies the character pair used to delimit keystroke parameters. Default is "[]" (square brackets).

Keystroke Parameters

key_char
An unquoted, case-sensitive literal character constant of length 1 representing a key on the keyboard which, when typed, represents a character visible on the display device. e.g. Z (upper case "z") represents keystroke shift-z. Therefore, if caps lock is on, shift-z would result in lower case "z" being typed.

Note that *key_char* may be blank representing a blank (space bar) keystroke. Therefore, blank characters may not be used to delimit keystrokes that are not enclosed by keystroke delimiter characters. e.g. `"A [TIMES 3]"` represents shift-a followed by 3 space bar key depressions.

Since *key_char* is a single character, it must not be enclosed by keystroke delimiter characters.

[*key_special***]**
A keystroke may be a key represented by an unquoted, case-insensitive literal character constant which has a length greater than 1 (e.g. [ENTER] or [enter] represents the Enter key). Because *key_special* is longer than 1 character, it must be enclosed in keystroke delimiter characters.

key_special is a Windows virtual-key code name with the "VK_" prefix omitted. Most, but not all, of these key codes are supported by SELCOPY. See Microsoft Developer Network documentation on "Virtual Key Codes" at:

`http://msdn.microsoft.com/en-us/library/windows/desktop/dd375731(v=vs.85).aspx`

Cursor movement is achieved with `[LEFT]`, `[RIGHT]`, `[UP]` and `[DOWN]`, and PFkeys may be specified as `[F1]`, `[F23]`, etc. For convenience, SELCOPY also supports the following *key_char* synonyms and abbreviations:

[RETURN]	[CR] [ENTER]
[LEFT]	[LE]
[RIGHT]	[RI]
[DOWN]	[DO] [DN]
[MENU]	[ALT]
[CONTROL]	[CTL] [CNTL] [CTRL]

[*key_combo***]**
Represents a *key_char* or *key_special* keystroke in combination with one of the special [ALT], [CTL] or [SHIFT] key modifiers.

To do this, the *key_char* or *key_special* keystroke value must be specified immediately before the closing keystroke delimiter character of [ALT], [CTL] or [SHIFT]. Since [ALT], [CTL] and [SHIFT] are also *key_special* keystrokes, they too can be modified by another key modifier, thus simulating a keystroke involving more than one key modifier. e.g. Ctrl+Alt+Z)

[ALT x]	alt-x
[ALT [F1]]	alt-F1
[CTL [ALT z]]	ctrl-alt-z
[CTL [SHIFT [F9]]]	ctrl-shift-F9

[TIMES=*int***]**
[X=*int***]**
TIMES (or X) specifies decimal integer value, *int*, and is used to determine the number of times the preceding keystroke (or keystroke combination) is to be performed. e.g.

```
WRITE KEYO  WIN="Title"  "abc[times 7]def [x 5]g"    * "abccccccdef     g".
WRITE KEYO               "[left][times 26]"          * Cursor left 26 times.
```

[INTERVAL=*int***]**
[INTVL=*int***]**
[DELAY=*int***]**
INTERVAL (or INTVL or DELAY) specifies decimal integer value, *int*, and may used to override the value specified by the WRITE parameter, INTERVAL.

Like the WRITE parameter, *int* specifies the number of milli-seconds that must elapse before the next keystroke (or keystroke combination) is performed.

This time interval will take effect only after INTERVAL has been processed within the keystroke sequence order. Thereafter, the interval will apply to all subsequent keystrokes performed by the current, or subsequent, WRITE operations for *fname*. The interval value may be further changed by an INTERVAL parameter on a subsequent WRITE operation or by another [INTERVAL=*int*] specification.

[WAIT=*int***]**
WAIT specifies decimal integer value, *int*, a one-off number of milli-seconds for which SELCOPY is to pause before continuing to process the remainder of the keystroke sequence.

If *int* exceeds 60000 (60 seconds) the requested WAIT is ignored. To pause SELCOPY processing for this length of time or longer, the SLEEP operation should be used.

Non-US and Non-UK National Keyboards

When SELCOPY is instructed to write a character as a keystroke to a different window, it translates that character to a Virtual Key Code before passing it to a Microsoft Windows function to actually type it at the cursor position of the target window.

By default, SELCOPY assumes that the destination window has a Standard US Keyboard Layout, which would result in the following special characters being wrongly interpreted if a non-US keyboard layout is in use (e.g. if a Standard Spanish Keyboard Layout is in effect):

Character	ASCII Hex	Description
"	x'22'	Double quote
#	x'23'	Hash
@	x'40'	Commercial at sign
`	x'60'	Backwards quote
~	x'7E'	Tilde
£	x'A3'	Currency symbol
¬	x'AC'	NOT sign

Conveniently, Microsoft Windows provides a method of cycling through the installed national keyboards which, in addition to the local region keyboard, usually includes **English-US**. Pressing the Alt+Shift keys combination will switch to the next, installed national keyboard.

For **English-UK** national keyboards, SELCOPY automatically performs an implied `[ALT [SHIFT]]` prior to processing keystroke output, and once again when keystroke processing is complete and so no special action is required.

For all other non-US national keyboards, the *key_combo*, `[ALT [SHIFT]]` must be specifically included in the keystroke string to temporarily switch the national keyboard of the target window to English-US. The following sample control statement file demonstrates this switch to an English-US keyboard and back again:

```
* "Non_US_test.ctl" is a file on the current directory of a m/c with a non-US keyboard.
*  ---------------
  system 'start notepad'    * No filename, so TitleBar will be "Untitled - Notepad".
  sleep 1                   * Wait 1 second, although unnecessary.
*          The notepad window will have a local keyboard layout.

  wr win='untitled'   delay 20  \
    '[cr]2013/06/20 20.14   "QUOT2"  @AT@  #HASH#  in non-US mode.[cr]'  * Problems.


  wr win='untitled'    '[ALT [SHIFT]]'   * To toggle the US/local keyboard layout.
                                         * Will result in US layout.
  wr win='untitled'   delay 5   \
    '[cr]2013/06/20 20.20   "QUOT2"  @AT@  #HASH#  after switch to US mode.[cr]'

  wr win='untitled'    '[ALT [SHIFT]]'   * To toggle the US/local keyboard layout.
                                         * Will result in restoring local layout.

  wr win RESET                           * To return focus to the window which
                                         * issued the command to run SELCOPY.
```

Executing SELCOPY against this control statement file will open the Notepad application for a new, "untitled" document and update it as follows:

```
2013/06/20 20.14   ~QUOT2~  "AT"  £HASH£  in non-US mode.

2013/06/20 20.20   "QUOT2"  @AT@  #HASH#  after switch to US mode.
```

Note that, since SELCOPY simulates keyboard input as if typed by a user at the target window, the Alt+Shift must be performed at the target window and not at the window in which SELCOPY executes. The national keyboard used in the SELCOPY execution window is irrelevant since the source of the keystroke string being processed is not via a keyboard but exists in SELCOPY's storage as data.

Keystroke Example

The following example demonstrates syntax that starts a web browser pointing at a search engine page, and then types and enters text at the search field prompt .

```
selcopy  !system 'start FIREFOX www.google.co.uk' !sleep 10  !write win=goo 'tn3270[cr]'  !end
```

This same syntax may be implemented as a more permanent solution by saving it in a batch command file script. The first parameter identifies the topic to be searched. e.g. a file "google.cmd" may be added to a folder in the search PATH containing the following:

```
:  c:\djh\ca\google.cmd **         L=001 --- 2013/06/16 21:53:06  (L07)
@echo off
:           Invoke a google search.
: L=001 2013/06/16 -djh- Use SLC to enter topic into google's topic box.
  if .%1==.   (
    echo google.cmd: Param 1 reqd to define topic.
    goto end  )

  start    FIREFOX www.google.co.uk
  selcopy  !option noban   !sleep 10  !write win=goo "%1[CR]"  !end
:end
```

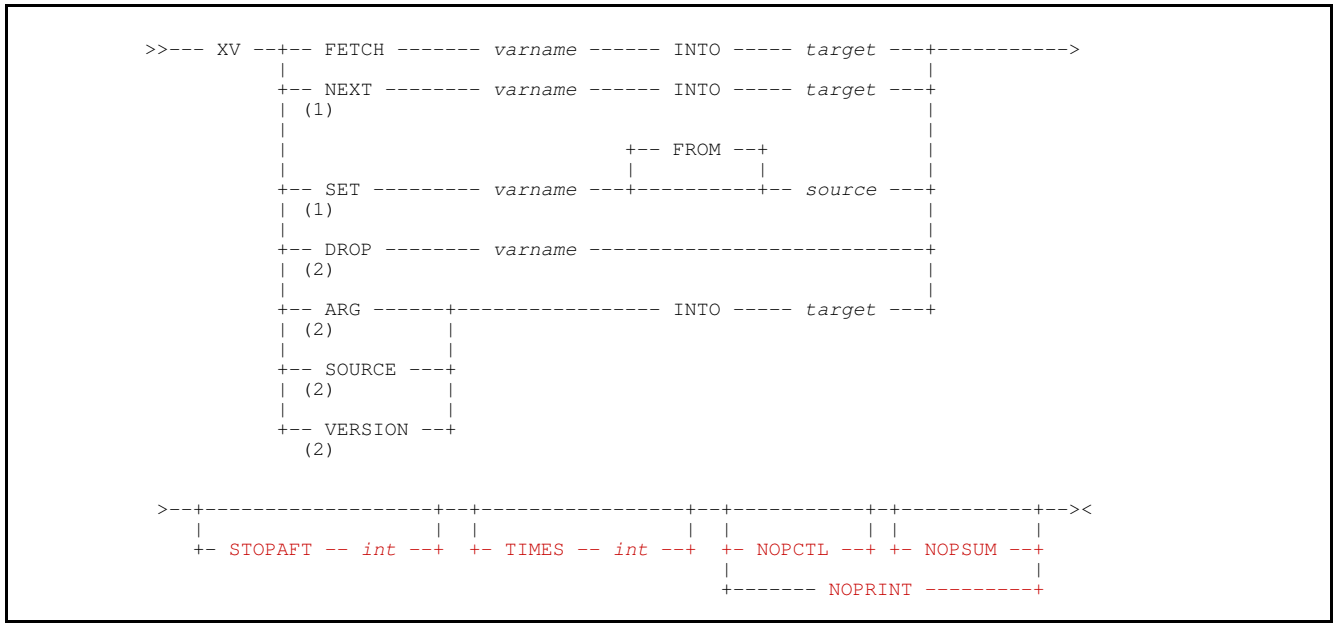
Thereafter, to perform a Google search, simply enter "google" followed by the search text at a Windows terminal command prompt or in the "Run" dialog. e.g.

```
google tn3270
```

XV

Transfer the value of a variable between the controlling environment and a SELCOPY declared variable or field definition.

Syntax:



Syntax Notes:

(1) NEXT and SET are applicable in z/OS and z/VM CMS REXX environments only. i.e. Not applicable in a Microsoft Windows or POSIX environment.
SET, but not NEXT, is also supported for CMS EXEC2 variables.

(2) DROP, ARG, SOURCE and VERSION are applicable in a z/VM CMS REXX environment only. i.e. Not applicable in a z/VM EXEC2, z/OS, Microsoft Windows or POSIX environment.

Description:

The XV operation provides a method of extracting, or updating the value of a variable from the controlling procedure/environment. The function keywords FETCH (GET), NEXT, SET, DROP ARG, SOURCE and VERSION dictate the action to be performed by the XV operation.

For Microsoft Windows and POSIX environments, only the FETCH function is supported for extracting the value of an environment variable. The value fetched is the same as would be obtained by an environment variable reference. e.g. %PATH%

In a z/OS REXX environment, the FETCH, NEXT and SET functions are supported.
In a z/VM CMS REXX environment, all XV functions are supported.
In a z/VM CMS EXEC2 environment, only the FETCH and SET functions are supported and reference to the EXEC2 variable name, *varname*, must not include the "&" (ampersand) prefix.

The return code value set by the XV operation is assigned to the internal variable RETSYS. e.g. In z/VM CMS, the REXX return code passed back to SELCOPY for the XV FETCH NEXT statement is as documented in IBM's "z/VM CMS Callable Services Reference" manual for the routine, DMSCGX. Some of the more common return code values set by this function and assigned to RETSYS include:

0	Normal completion.
200	The data assigned to *target* and/or *varname* has been truncated. SELCOPY will also set its own RC=8.
202	REXX is not active.
206	All variables have been processed.
207	Unsupported function. (This return code is issued if DMSCGX is called from within an EXEC2 environment.)

Parameters:

FETCH
GET
Fetch the value of the environment variable referenced by *varname* and assign it to *target*. If the varname is a stem of a REXX variable, the initial value of that stem (if any) is returned.

NEXT
Applicable only in a (z/OS or z/VM CMS) REXX environment, the NEXT function will fetch the name and value of the next variable from the stack of variables known to the language processor. (i.e. all those of the current generation, excluding those hidden by PROCEDURE instructions). The order in which the variables are revealed is not specified.

The variable name will be assigned to *varname* and the *value* to target. Therefore, for the NEXT function, *varname* cannot be specified as a character constant.

By repeatedly executing the NEXT function, the SELCOPY program can obtain all the REXX variables of the current generation.

SET
Applicable only in a (z/OS or z/VM CMS) REXX or (z/VM CMS) EXEC2 environment, the SET function will set the value of a new or existing variable. If the name is a REXX variable stem, then all variables with that stem are set.

DROP
Applicable only in a z/VM CMS REXX environment, the DROP function will drop the REXX variable specified by *varname*, if it exists. If the name given is a REXX variable stem, then all variables starting with that stem are dropped.

ARG
Applicable only in a z/VM CMS REXX environment, the ARG function will fetch the primary argument string and assign it to *target*. i.e. The first argument string passed to the REXX procedure that would be parsed by the PARSE ARG instruction.

SOURCE
Applicable only in a z/VM CMS REXX environment, the SOURCE function will fetch the source string and assign it to *target*. i.e. the string that would be returned by the PARSE SOURCE instruction.

VERSION
Applicable only in a z/VM CMS REXX environment, the VERSION function will fetch the version string and assign it to *target*. i.e. The string that would be returned by the PARSE VERSION instruction.

varname
If the function is FETCH, SET or DROP, *varname* identifies the name of the variable for which the value will be fetched, set or dropped. For these functions, *varname* may be specified as an unquoted literal, a declared variable of character data type or a Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

If the function is NEXT, then *varname* identifies the target to which the next variable name will be assigned. For this function, *varname* may be specified as a declared variable or field definition only. (i.e. not an unquoted literal.)

source
For the function SET only, *source* identifies the source from which the variable's value will be assigned. *source* may be specified as a quoted character constant, a declared variable of character data type or a Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

target
For the functions FETCH, NEXT, ARG, SOURCE and VERSION only, *target* identifies the target to which the variable's value will be assigned. *target* may be specified as a declared variable of character data type or a Type 2 (*field_p1p2*) or Type 3 (*field_nATp*) field definition.

NOPCTL, NOPSUM, NOPRINT
See common parameters NOPCTL, NOPSUM, NOPRINT for details.

STOPAFT *int*
See common parameter STOPAFT for details.

TIMES *int*
See common parameter TIMES for details.

Example:

The following sample output is from a SELCOPY program executed from a z/VM CMS REXX procedure that uses the XV NEXT operation to report all current generation REXX variables and their values.

```
SELCOPY/CMS 3.30 at CBL - Bridgend UK (Internal Only)              2016/01/11 16:48   PAGE   1
-----------------------------------------------------              ----------------   --------


          DECLARE  VNAME   CHAR(15)
          DECLARE  VVALUE  CHAR(40)

        ==VARLOOP==
          -------
     1.   XV   NEXT   VNAME   INTO VVALUE

          IF RETSYS=0
     2.     THEN PLOG TYPE=C 'Variable Name: ' VNAME '  Value: ' VVALUE
     3.     THEN GOTO VARLOOP


   INPUT   SEL SEL                                                                                  1  RECORD
   RECNO   TOT ID.          1         2         3         4         5         6         7         8         9         0  LENGTH
   -----   --- --- ....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0....,....0  ------
       0     1   2 Variable Name: RC               Value: 8                                                                80
       0     2   2 Variable Name: XPARM.NO         Value: FAILURE                                                          80
       0     3   2 Variable Name: XPARM.UNKNOWN    Value: Not Found                                                        80
       0     4   2 Variable Name: XPARM.OK         Value: DJH                                                              80
       0     5   2 Variable Name: JGE              Value: OK                                                               80
                   ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,....0

SUMMARY..
 SEL-ID      SELTOT      FILE     BLKSIZE  LRECL           FSIZE   CI    DSN
 ------      ------      ----     -------  -----           -----   --    ---

                   =VARLOOP=
   1             6
   2----3        5


   ** SELCOPY/CMS 3.30.001  Licensed by Compute (Bridgend) Ltd  +44 (1656) 652222 & 656466 **
                                  ** Expiry: 14 Jul 2016 **
```

Figure 26. XV NEXT operation.

Appendix A. Regular Expression Summary

The following tables are a summary of all regular expression operators, text specifiers and predefined expressions.

Operators and Text Specifiers

Operator	Description
\	Escape the next char, treat as text, except for certain standard ESC sequences: `\a` Apostrophe (Single Quote) (Not yet implemented.) `\q` Double Quote (Not yet implemented.) `\b BS` BackSpace ASCII X'08' EBCDIC X'16' `\f FF` FormFeed ASCII X'0C' EBCDIC X'0C' `\r CR` Carriage Return ASCII X'0D' EBCDIC X'0D' `\l NL` New Line EBCDIC X'15' (Not yet implemented.) `\n LF` LineFeed ASCII X'0A' EBCDIC X'25' `\t HT` Horizontal Tab ASCII X'09' EBCDIC X'05' `\xnn` Hexadecimal character code X'xx' `\nnn` Octal character code nnn e.g. \072 or \72 (equivalent to \x3A)
^	Must be at POS 1 of source data field if '^' is 1st character of RGX.
$	Must have exhaused the source data field.
[class]	Matches any character in enclosed class.
[~class]	Matches any character NOT in enclosed class. The '~' must be 1st in class.
[a-z]	Matches any character within the range 'a' to 'z', or any other range.
( )	Groups any enclosed RGX to be treated as a unit.
(x1|x2|...)	Alternation where any one of the | separated terms gives a match.
Repetition Operators	
*	0 Min Closure - match prev term 0 or as many times as reqd.
+	1 Min Plus - match prev term 1 or as many times as reqd.
@	0 Max Closure - match prev term 0 or as many times as Possible.
#	1 Max Plus - match prev term 1 or as many times as Possible.
^n	Power factor - match prev term n times precisely, if '^' not 1st character of RGX.
~	Not function - succeeds only if next RGX term is NOT matched. Advances the RGX, but not the source data field.
{ }	Tagged expression enables reference, later in the RGX or in the substitution field of a CHANGE statement, to the data matched within the enclosing braces.
&n	References the data matched in the nth tagged expression
?	Wildcard character - matches any single character
:	Treats the colon and the next character as shorthand for a Predefined Expression. The next character must be one of the lower case letters: a b c d q w z

Predefined Expressions

Name	Definition	Description
:a	`[a-zA-Z0-9]`	Alphanumeric character
:b	`([\t ]#)`	White space (a string of blanks and tabs).
:c	`[a-zA-Z]`	Alphabetic character
:d	`[0-9]`	Numeric digit
:q	`(("[~"]@")|('[~']@'))`	Quoted string (in single or double quotes).
:w	`([a-zA-Z]#)`	Word (a string of alphabetic characters).
:z	`([0-9]#)`	Integer (a string of numeric digits).

Appendix B. Operation, Parameter and Argument Keywords

The following tables are a cross reference of all operation, parameter and argument keywords supported by SELCOPY.

Operation to Parameter Keyword Cross-Reference

Operation	Parameters					
ADD	INTO	TO				
ADJC	See **CENTRE**					
ADJL	See **LEFT**					
ADJR	See **RIGHT**					
CALL	No specific parameter keywords.					
CENTER	See **CENTRE**					
CENTRE	FROM	TO				
CHANGE	CASEI	FILL	HITS	NULL	FILL	
CHOP	No specific parameter keywords.					
CLOSE	APPEND	BDW	BLKSIZE	DD	DEFER	DIR
	DIRDATA	DIRTYPE	DSN	DSNPFX	EOL	ESDS
	FILL	FMT	HEADER	KEYLEN	KEYPOS	KSDS
	LIST	LRECL	NOBDW	NORDW	NOTRUNC	ODBCPASS
	RAW	RDW	RECFM	REUSE	RRDS	SELECT
	SEP	SORT	SORTDIR	SQL	SSN	SUBDIR
	TABLE	TABS	TRUNC	UPD	USER	VSAM
	WHERE	WIN				
COMPRESS	ALL	DLM	ENC	ESC	FILL	FLEN
	FROM	IFNEC	PAD	STR	TO	
CP	FORMAT	FROM	INTO	REPLY		
CVA	See **CVXX** (CVEA)					
CVB	See **CVXX** (CVPB)					
CVD	See **CVXX** (CVBP)					
CVE	See **CVXX** (CVAE)					
CVH	See **CVXX** (CVCH)					
CVDATE	DATECB	FROM	NOW	TO		
CVXX	FROM	TO				
DECLARE	BIN	CHAR	CHARVARYING	CHARZ	CSTRING	DEC
	DOUBLE	FLOAT	FORMAT	FP	INIT	NTS
	POS	VARCHAR	VHAR	ZCHAR		
DELETE	DSN	FILE	TABLE			
DIVIDE	BY	INTO	REM			
DO	FORMAT	FROM				
DUMMY	No specific parameter keywords.					
END	No specific parameter keywords.					
EQU	No specific parameter keywords.					
EXPAND	DELIM	ENC	ESC	FILL	FLEN	FROM
	PAD	TO				
EOJ	See **GOTO EOJ**					
FLAG	EODIR	EODISK	EOMBR	EOMEMB	FILE	
FLUSH	FILE					
GENERATE	BASE	RANGE				
GET	See **READ**					
GOSUB	See **DO**					
GOTO	CANCEL	EOJ	GET			
IF/AND/OR	AND	ANY	CASEI	DATA	DIR	ELSE
	EOF	FILE	FILL	INCOUNT	LINE	MATCHLEN
	NOT	NULL	NUMERIC	OR	PAD	POS
	PTR	REVERSE	STEP	THEN		

IN	See **READ**					
INCLUDE	No specific parameter keywords.					
INPUT	See **READ**					
INSERT	DSN	FILE	FORMAT	FROM		
LEFT	FROM	TO				
LOG	DATAWIDTH	DUMPALL	DUMPENC	FORMAT	FROM	INTO
	PAGEWIDTH	REPLY	TYPE			
LOWER	FROM	TO				
MOD	FILL	FORMAT	FROM	MOD	PAD	
MOVE	FILL	FROM	PAD	TO		
MULTIPLY	BY	INTO				
ODBC	ODBCPASS	PASSWORD	SSN	USER		
OPEN	APPEND	BDW	BLKSIZE	DD	DEFER	DIR
	DIRDATA	DIRTYPE	DSN	DSNPFX	EOL	ESDS
	FILL	FMT	HEADER	KEYLEN	KEYPOS	KSDS
	LIST	LRECL	NOBDW	NORDW	NOTRUNC	ODBCPASS
	RAW	RDW	RECFM	REUSE	RRDS	SELECT
	SEP	SORT	SORTDIR	SQL	SSN	SUBDIR
	TABLE	TABS	TRUNC	UPD	USER	VSAM
	WHERE	WIN				
OPTION	ABTRAP	BANNER	CALLTYPE	CBLSQLOG	DATAWIDTH	DEFAULTDIR
	DEFAULTFP	DIRTYPE	DSNPFX	DUMPALL	DUMPENC	ENVFAIL
	ENVVAR	ERRLIM	ERRMAX	FILL	HEAD	LIBNAME
	LOGSQL	MFC	NOBANNER	NOENVVAR	NOPCTL	NOPRINT
	NOPSUM	NOPTOT	NORDW	NOSUB	NOTRAP	ODBCPASS
	PAD	PAGEDEPTH	PAGEWIDTH	PASS	PRINTABLE	PRRECLEN
	PRTCTL	PRTSUM	RANGE	RC_KEYNF	RDW	REPORT
	SEP	SITE	SORTDIR	SQLOG	SSN	SUBDIR
	SUMPRT	TRAP	TYPEDIR	UNPRINTABLE	USER	USERID
	WORKAREA	WORKLEN				
PACK	See **CVXX** (CVCP)					
PACKB	See **CVXX** (CVCB)					
PARSE	See **CHOP**					
PERFORM	See **DO**					
PLOG	DATAWIDTH	DUMPALL	DUMPENC	FORMAT	FROM	PAGEDEPTH
	PAGEWIDTH	TYPE				
PRINT	DATAWIDTH	DUMPALL	DUMPENC	FORMAT	FROM	PAGEDEPTH
	PAGEWIDTH	TYPE				
RANDOM	See **GENERATE**					
READ	ALTX	BDW	BLKSIZE	BWD	DD	DEFER
	DIR	DIRDATA	DIRTYPE	DSN	DSNPFX	EOL
	ESDS	FILE	FILL	FMT	FWD	HEADER
	INTO	KEQ	KEY	KEYIX	KEYLEN	KEYPOS
	KGE	KSDS	LIST	LRECL	NOBDW	NODSNPFX
	NORDW	NOUSERID	ODBCPASS	PAD	PASSWORD	RAW
	RBA	RDW	REC	RECFM	RECSIZE	RRDS
	SELECT	SEP	SORT	SORTDIR	SQL	SSN
	STARTKEY	STARTKGE	STARTRBA	STARTREC	STOPAFT	SUBDIR
	TABLE	TABS	TYPE	TYPEDIR	UPD	USER
	USERID	VSAM	WHERE	WORKAREA	WORKLEN	WTO
RETURN	No specific parameter keywords.					
RIGHT	FROM	TO				
SLEEP	HOURS	MINS	SECS			
SPACE	No specific parameter keywords.					
STACK	EOF	FIFO	FORMAT	FROM	LIFO	NULL
START	No specific parameter keywords.					
STOP	See **GOTO EOJ**					
SUB	FROM	INTO				
SUSP	No specific parameter keywords.					

SYSTEM	FORMAT	FROM				
TRAN	ASCII	EBCDIC	FROM	HITS	LOWER	PRINT
	TAB	TO	UPPER			
UNPACK	See **CVXX** (CVPC)					
UNPACKB	See **CVXX** (CVBC)					
UPDATE	DSN	FILE	FORMAT	FROM	TABLE	
UPPER	FROM	TO				
UTIME	DSN	FILE	FTIME			
WRITE	APPEND	BDW	BLKSIZE	DEFER	DELAY	DSN
	DSNPFX	EOL	ESDS	FILE	FILL	FMT
	FORMAT	FROM	INTERVAL	KEYENC	KEYENCERR	KEYLEN
	KEYPOS	KSDS	LRECL	NEWBLK	NOBDW	NODSNPFX
	NOKEYENC	NOKEYENCERR	NOTRUNC	NOUSERID	ODBCPASS	PAD
	PASSWORD	RECFM	RECSIZE	RRDS	SSN	TABLE
	TRUNC	USER	USERID	VSAM	WIN	WTO
XV	ARG	DROP	FETCH	GET	NEXT	SET
	SOURCE	VERSION				

Parameter to Operation Keyword Cross-Reference

Parameter	Operation(s)					
ABTRAP	OPTION					
ALL	COMPRESS					
ALTX	READ					
AND	IF/AND/OR					
ANY	IF/AND/OR					
APPEND	CLOSE	OPEN	WRITE			
ARG	XV					
ASCII	TRAN					
AT	Field Definitions					
BANNER	OPTION					
BASE	GENERATE					
BDW	CLOSE	OPEN	READ	WRITE		
BIN	DECLARE					
BLKSIZE	CLOSE	OPEN	READ	WRITE		
BWD	READ					
BY	DIVIDE	MULTIPLY				
CALLTYPE	OPTION					
CANCEL	GOTO					
CASEI	CHANGE	IF/AND/OR				
CBLSQLOG	OPTION					
CHAR	DECLARE					
CHARVARYING	DECLARE					
CHARZ	DECLARE					
CSTRING	DECLARE					
DATA	IF/AND/OR					
DATAWIDTH	LOG	OPTION	PLOG	PRINT		
DATECB	CVDATE					
DD	CLOSE	OPEN	READ			
DEC	DECLARE					
DEFAULTDIR	OPTION					
DEFAULTFP	OPTION					
DEFER	CLOSE	OPEN	READ	WRITE		
DELAY	WRITE					
DELIM	EXPAND					
DIR	CLOSE	IF/AND/OR	OPEN	READ		
DIRDATA	CLOSE	OPEN	READ			

DIRTYPE	CLOSE	OPEN	OPTION	READ		
DLM	COMPRESS					
DOUBLE	DECLARE					
DROP	XV					
DSN	CLOSE	DELETE	INSERT	OPEN	READ	UPDATE
	UTIME	WRITE				
DSNPFX	CLOSE	OPEN	OPTION	READ	WRITE	
DUMPALL	LOG	OPTION	PLOG	PRINT		
DUMPENC	LOG	OPTION	PLOG	PRINT		
EBCDIC	TRAN					
ELSE	IF/AND/OR					
ENC	COMPRESS	EXPAND				
ENVFAIL	OPTION					
ENVVAR	OPTION					
EODIR	FLAG					
EODISK	FLAG					
EOF	STACK	IF/AND/OR				
EOJ	GOTO					
EOL	CLOSE	OPEN	READ	WRITE		
EOMBR	FLAG					
EOMEMB	FLAG					
ERRLIM	OPTION					
ERRMAX	OPTION					
ESC	COMPRESS	EXPAND				
ESDS	CLOSE	OPEN	READ	WRITE		
FETCH	XV					
FIFO	STACK					
FILE	DELETE	FLAG	FLUSH	IF/AND/OR	INSERT	READ
	UPDATE	UTIME	WRITE			
FILL	CHANGE	CLOSE	COMPRESS	EXPAND	IF/AND/OR	MOD
	MOVE	OPEN	OPTION	READ	WRITE	
FLEN	COMPRESS	EXPAND				
FLOAT	DECLARE					
FMT	CLOSE	OPEN	READ	WRITE		
FORMAT	CP	DECLARE	DO	Field Definitions	INSERT	LOG
	MOD	PLOG	PRINT	STACK	SYSTEM	UPDATE
	WRITE					
FP	DECLARE					
FROM	CENTRE	COMPRESS	CP	CVDATE	CVxx	DO
	EXPAND	Field Definitions	INSERT	LEFT	LOG	LOWER
	MOD	MOVE	PLOG	PRINT	RIGHT	STACK
	SUB	SYSTEM	TRAN	UPDATE	UPPER	WRITE
FTIME	UTIME					
FWD	READ					
GET	GOTO	XV				
HEAD	OPTION					
HEADER	CLOSE	OPEN	READ			
HITS	CHANGE	TRAN				
HOURS	SLEEP					
IFNEC	COMPRESS					
INCOUNT	IF/AND/OR					
INIT	DECLARE					
INTERVAL	WRITE					
INTO	ADD	CHOP	CP	DIVIDE	LOG	MULTIPLY
	READ	SUB				
KEQ	READ					
KEY	READ					
KEYENC	WRITE					
KEYENCERR	WRITE					

KEYIX	READ					
KEYLEN	CLOSE	OPEN	READ	WRITE		
KEYPOS	CLOSE	OPEN	READ	WRITE		
KGE	READ					
KSDS	CLOSE	OPEN	READ	WRITE		
LENGTH	Field Definitions					
LIBNAME	OPTION					
LIFO	STACK					
LINE	IF/AND/OR					
LIST	CLOSE	OPEN	READ			
LOGSQL	OPTION					
LOWER	TRAN					
LRECL	CLOSE	OPEN	READ	WRITE		
MATCHLEN	IF/AND/OR					
MFC	OPTION					
MINS	SLEEP					
MOD	MOD					
NEWBLK	WRITE					
NEXT	XV					
NOBANNER	OPTION					
NOBDW	CLOSE	OPEN	READ	WRITE		
NODSNPFX	READ	WRITE				
NOENVVAR	OPTION					
NOKEYENC	WRITE					
NOKEYENCERR	WRITE					
NOPCTL	Common parm	OPTION				
NOPRINT	Common parm	OPTION				
NOPSUM	Common parm	OPTION				
NOPTOT	Common parm	OPTION				
NORDW	CLOSE	OPEN	OPTION	READ		
NOSUB	OPTION					
NOT	IF/AND/OR					
NOTRAP	OPTION					
NOTRUNC	CLOSE	OPEN	WRITE			
NOUSERID	READ	WRITE				
NOW	CVDATE					
NTS	DECLARE					
NULL	CHANGE	IF/AND/OR	STACK			
NUMERIC	IF/AND/OR					
ODBCPASS	CLOSE	ODBC	OPEN	OPTION	READ	WRITE
OR	IF/AND/OR					
PAD	CHANGE	COMPRESS	EXPAND	IF/AND/OR	MOD	MOVE
	OPTION	READ	WRITE			
PAGEDEPTH	OPTION	PLOG	PRINT			
PAGEWIDTH	LOG	OPTION	PLOG	PRINT		
PASS	OPTION					
PASSWORD	ODBC	READ	WRITE			
POS	DECLARE	Field Definitions	IF/AND/OR			
PRINT	TRAN					
PRINTABLE	OPTION					
PRRECLEN	OPTION					
PRTCTL	OPTION					
PRTSUM	OPTION					
PTR	IF/AND/OR					
RANGE	GENERATE	OPTION				
RAW	CLOSE	OPEN	READ			
RBA	READ					
RC_KEYNF	OPTION					
RDW	CLOSE	OPEN	OPTION	READ		

REC	READ					
RECFM	CLOSE	OPEN	READ	WRITE		
RECSIZE	READ	WRITE				
REM	DIVIDE					
REPLY	CP	LOG				
REPORT	OPTION					
REUSE	CLOSE	OPEN				
REVERSE	IF/AND/OR					
RRDS	CLOSE	OPEN	READ	WRITE		
SECS	SLEEP					
SELECT	CLOSE	OPEN	READ			
SEP	CLOSE	OPEN	OPTION	READ		
SET	XV					
SITE	OPTION					
SORT	CLOSE	OPEN	READ			
SORTDIR	CLOSE	OPEN	OPTION	READ		
SOURCE	XV					
SQL	CLOSE	OPEN	READ			
SQLOG	OPTION					
SSN	CLOSE	ODBC	OPEN	OPTION	READ	WRITE
STARTKEY	READ					
STARTKGE	READ					
STARTRBA	READ					
STARTREC	READ					
STEP	IF/AND/OR					
STOPAFT	READ	Common parm				
STR	COMPRESS					
STYLE	Field Definitions					
SUBDIR	CLOSE	OPEN	OPTION	READ		
SUMPRT	OPTION					
TAB	TRAN					
TABLE	CLOSE	DELETE	OPEN	READ	UPDATE	WRITE
TABS	CLOSE	OPEN	READ			
THEN	IF/AND/OR					
TIMES	Common parm					
TO	ADD	CENTRE	COMPRESS	CVDATE	CVxx	EXPAND
	LEFT	LOWER	MOVE	RIGHT	TRAN	UPPER
TRAP	OPTION					
TRUNC	CLOSE	OPEN	WRITE			
TYPE	Field Definitions	LOG	PLOG	PRINT	READ	
TYPEDIR	OPTION	READ				
UNPRINTABLE	OPTION					
UPD	CLOSE	OPEN	READ			
UPPER	TRAN					
USER	CLOSE	ODBC	OPEN	OPTION	READ	WRITE
USERID	OPTION	READ	WRITE			
VARCHAR	DECLARE					
VERSION	XV					
VHAR	DECLARE					
VSAM	CLOSE	OPEN	READ	WRITE		
WHERE	CLOSE	OPEN	READ			
WIN	CLOSE	OPEN	WRITE			
WORKAREA	OPTION	READ				
WORKLEN	OPTION	READ				
WTO	READ	WRITE				
ZCHAR	DECLARE					

Parameter to Argument Keyword Cross-Reference

Parameter	Argument(s)					
ABTRAP	OFF	ON				
CALLTYPE	DIRECT	VIA_SLCCALL				
DEFAULTFP	BFP	BIN	HEX	HFP	NATIVE	
DSNPFX	NO	YES				
DUMPALL	NO					
EOL	CR	CRLF	LF	NO		
FILE	CARD	CLIP	DUMMY	STDIN (Input only)	STDOUT (Output only)	SYSIN (Input only)
FLOAT	BFP	BIN	HEX	HFP	NATIVE	
HEAD	NO					
KEYENC	NO	OFF				
KEYENCERR	NO	OFF	ON	YES		
PRRECLEN	NO	YES				
RECFM	F	FB	MFV	U	V	V2
	V3	VB				
SEP	NO					
SORTDIR	0	D	E	N	NO	P
	S					
STYLE	A	B	D	I	J	T
TRAP	OFF	ON				
TYPE (Field)	B	C	F	P	U	Z
TYPE=F	BFP	BIN	HEX	HFP	NATIVE	
TYPE (Print)	B	C	D	DX	H	M
	MC	MP	N	S		
WIN	RESET					

Argument to Parameter Keyword Cross-Reference

Argument	Parameter(s)					
0	SORTDIR					
A	STYLE					
B	STYLE	TYPE (Field)	TYPE (Print)			
BFP	DEFAULTFP	TYPE=F	FLOAT			
BIN	DEFAULTFP	TYPE=F	FLOAT			
C	TYPE (Field)	TYPE (Print)				
CARD	FILE					
CLIP	FILE					
CR	EOL					
CRLF	EOL					
D	SORTDIR	STYLE	TYPE (Print)			
DIRECT	CALLTYPE					
DUMMY	FILE					
DX	TYPE (Print)					
E	SORTDIR					
F	RECFM	TYPE (Field)				
FB	RECFM					
H	TYPE (Print)					
HEX	DEFAULTFP	TYPE=F	FLOAT			
HFP	DEFAULTFP	TYPE=F	FLOAT			
I	STYLE					
J	STYLE					
LF	EOL					
M	TYPE (Print)					

MC	TYPE (Print)					
MFV	RECFM					
MP	TYPE (Print)					
N	SORTDIR	TYPE (Print)				
NATIVE	DEFAULTFP	TYPE=F	FLOAT			
NO	DSNPFX	DUMPALL	KEYENCERR	PRRECLEN	KEYENC	HEAD
	SEP	SORTDIR	EOL			
OFF	ABTRAP	TRAP	KEYENCERR	KEYENC		
ON	ABTRAP	TRAP	KEYENCERR			
P	SORTDIR	TYPE (Field)				
RESET	WIN					
S	SORTDIR	TYPE (Print)				
STDIN	FILE (Input)					
STDOUT	FILE (Output)					
SYSIN	FILE (Input)					
T	STYLE					
U	RECFM	TYPE (Field)				
V	RECFM					
V2	RECFM					
V3	RECFM					
VB	RECFM					
VIA_SLCCALL	CALLTYPE					
YES	DSNPFX	DUMPALL	KEYENCERR	PRRECLEN		
Z	TYPE (Field)					

Keyword Abbreviations

The following table identifies the supported keyword abbreviations for SELCOPY operation, sub-operation, parameter, argument and internal variable keywords.

Abbreviation	Type	Keyword
A	Sub-operation	AND
APP	Parameter	APPEND
B	Parameter	BIN
	Parameter	BLKSIZE
BAN	Parameter	BANNER
BLK	Parameter	BLKSIZE
C	Parameter	CHAR
	Operation	CHANGE
CH	Operation	CHANGE
CHA	Parameter	CHAR
CHARV	Parameter	CHARVARYING
CHAZ	Parameter	CHARZ
CHG	Operation	CHANGE
CHV	Parameter	CHARVARYING
CHZ	Parameter	CHARZ
CSTR	Parameter	CSTRING
CVDT	Operation	CVDATE
D	Parameter	DEC
DBL	Parameter	DOUBLE
DCL	Operation	DECLARE
DEFDIR	Parameter	DEFAULTDIR
DEFFP	Parameter	DEFAULTFP
DEL	Operation	DELETE
DFLTDIR	Parameter	DEFAULTDIR
DFLTFP	Parameter	DEFAULTFP
DIRD	Parameter	DIRDATA
DIV	Operation	DIVIDE

Abbreviation	Type	Keyword
DLET	Operation	DELETE
DLM	Parameter	DELIM
DW	Parameter	DATAWIDTH
E	Operation	END
EL	Sub-operation	ELSE
ELSEIF	Sub-operation	ELSE IF
EOD	Parameter	EODIR
EODSK	Parameter	EODISK
EOM	Parameter	EOMEMB
EOMEM	Parameter	EOMEMB
F	Parameter	FILE
	Parameter	FLOAT
FLT	Parameter	FLOAT
FMAT	Parameter	FORMAT
FMT	Parameter	FORMAT
FR	Parameter	FROM
GEN	Operation	GENERATE
H	Parameter	HEAD
	Parameter	HEADER
HD	Parameter	HEAD
	Parameter	HEADER
HDR	Parameter	HEADER
HEAD	Parameter	HEADER
I	Operation	IF
IN	Variable	INCOUNT
	Operation	INPUT
INC	Operation	INCLUDE

Abbreviation	Type	Keyword
INI	Parameter	INIT
INS	Operation	INSERT
INTVL	Parameter	INTERVAL
ISRT	Operation	INSERT
KL	Parameter	KEYLEN
KP	Parameter	KEYPOS
L	Sub-operation	ELSE
	Parameter	LENGTH
	Parameter	LRECL
	Variable	LRECL
LEN	Parameter	LENGTH
LI	Sub-operation	ELSE IF
MULT	Operation	MULTIPLY
NAT	Argument	NATIVE
NOBAN	Parameter	NOBANNER
NOP	Parameter	NOPRINT
NOUID	Parameter	NOUSERID
NUL	Parameter	NULL
NUM	Parameter	NUMERIC
O	Sub-operation	OR
OPASS	Parameter	ODBCPASS
OPT	Operation	OPTION
OPTIONS	Operation	OPTION
P	Parameter	POS
PASS	Parameter	PASSWORD
PASSWD	Parameter	PASSWORD
PD	Parameter	PAGEDEPTH
PR	Operation	PRINT
PRT	Operation	PRINT
PW	Parameter	PAGEWIDTH
R	Parameter	REPORT
RC	Variable	RETCODE
RD	Operation	READ

Abbreviation	Type	Keyword
REP	Operation	REPL
RET	Operation	RETURN
RETCD	Variable	RETCODE
REV	Parameter	REVERSE
S	Parameter	STOPAFT
SEL	Parameter	SELECT
SORT	Parameter	SORTDIR
START	Parameter	STARTKEY
STOP	Parameter	STOPAFT
SUB	Parameter	SUBDIR
SYS	Operation	SYSTEM
T	Sub-operation	THEN
TAB	Parameter	TABLE
THENIF	Sub-operation	THEN IF
TI	Sub-operation	THEN IF
TR	Operation	TRAN
TY	Parameter	TYPE
TYPE	Parameter	TYPEDIR
UID	Parameter	USERID
UPD	Operation	UPDATE
V	Parameter	VARCHAR
VC	Parameter	VARCHAR
VCH	Parameter	VARCHAR
VCHA	Parameter	VARCHAR
VCHAR	Parameter	VARCHAR
W	Parameter	WORKAREA
WORKA	Parameter	WORKAREA
WR	Operation	WRITE
Z	Parameter	ZCHAR
ZC	Parameter	ZCHAR
ZCH	Parameter	ZCHAR
ZCHA	Parameter	ZCHAR

Glossary

Base address
The base address is the address of the base storage area and is referenced in SELCOPY control statements as position 1 (POS 1). If the base storage area is defined by a work area, its address is static. Otherwise, its address changes with each record read from the prime input object.

Base storage area
The area of storage used as the work buffer. If no input object exists or option WORKLEN is specified, a work area is used as the base storage area. Otherwise, the base storage area is the position and length of the last record read from the prime input object.

Big-Endian
Big-endian describes the order of bytes that constitute a multi-byte data type so that the most significant value of the sequence is stored at the lowest storage address in computer memory. e.g. Decimal 100 expressed as a 4-byte, big-endian, binary constant is X'0000,0064'.

IBM z/Architecture (mainframe) hardware supports big-endian format only. Modern IBM POWER, Oracle SPARC and HP Alpha hardware are bi-endian and so can support either big-endian and little-endian formats natively.

Control statement analysis
The first step in SELCOPY execution which involves interpretation of statement elements, selection identifier assignment and data object open.

Little-Endian
Little-endian describes the order of bytes that constitute a multi-byte data type so that the least significant value of the sequence is stored at the lowest storage address in computer memory. e.g. Decimal 100 expressed as a 4-byte, little-endian, binary constant is X'6400,0000'.

Intel x86 and x86-64 architectures support little-endian format only.

Prime input
The input data object identified by the first READ operation encountered during control statement analysis. By default, end-of-job processing will only start when end-of-file (input data) occurs for the prime input data object. Also, the file name assigned to the prime input is default on operations where a required file name is omitted. e.g. IF EOF

Record
A generic term used to identify a length of data read from or written to a data object. e.g. If the data object is a database table (accessed via ODBC), then a record is a formatted table row.

Selection time processing
The second step in SELCOPY execution which involves execution of executable statements in a logical sequence. When the last executable statement in the sequence has been executed and if no input object exists, then end of job processing starts. However, if an input object exists, control is passed back to the first executable statement. This looping back to the first executable statement will occur until the end-of-file condition is encountered for the prime input object.

Work area
An allocated area of storage used as the base storage area with a static base address. By default, input data records are copied from the object's input buffer to position 1 of the work area. Similarly, output records are copied from position 1 of the work area to the output buffer.