NOS/VE
Advanced File Management

Usage

60486413

NOS/VE Advanced File Management

Usage

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features and parameters.

Publication Number 60486413

Manual History

Revisions	System Version/ PSR Level	Product Version	Date
A	1.0.2	1.0	October 1983
B	1.1.1/613	1.0	June 1984
C	1.1.2/630	1.0	March 1985
D	1.1.3/644	1.1	October 1985
E	1.1.4/649	1.2	January 1986
F	1.2.1/664	1.2/1.3	July 1986
G	1.2.2/678	1.3/1.4	April 1987
H	1.2.3/688	1.3/1.5	September 1987
J	1.3.1/700	1.3/1.6	April 1988

This revision:

Revision J documents Sort/Merge 1.3, the keyed-file utilities 1.6, and FMU 1.1 for NOS/VE Version 1.3.1 at PSR level 700. It was published in April 1988.

This revision documents the new NOS/VE command utility RESTORE_LOG, which is used to recover damaged log files. Also, minor technical and editorial changes have been made.

Contents

Figures

Tables

About This Manual

This manual describes three CONTROL DATA® System Command Language (SCL) file management tools for use under the Control Data's Network Operating System/Virtual Environment (NOS/VE). The three file management tools are Sort/Merge, the keyed-file utilities, and the File Management Utility (FMU).

Audience

This manual is written for any user of NOS/VE files who requires a means of sorting or reformatting records or uses keyed files (indexed-sequential or direct-access files).

The reader is assumed to be familiar with SCL command conventions, NOS/VE system access, and the NOS/VE file system. This information is given in the NOS/VE System Usage manual.

This manual is a usage manual, meaning that it contains a comprehensive description of how to use the software indicated in its title. For a tutorial that introduces you to the software described in this manual, see the SCL Advanced File Management Tutorial manual.

Manual Organization

This manual is divided into four parts as follows:

- The first part describes the SCL interface to Sort/Merge.
- The second part describes the keyed-file utilities.
- The third part describes FMU usage.
- The fourth part of this manual contains appendixes. The appendixes provide a glossary, character set and collating sequence listings, and a comparison of the NOS/VE products described in this manual and their predecessor products.

  Two appendixes in this part supplement the FMU descriptions. These are the FMU Messages appendix and the FMU Conversion Rules, Storage Requirements, and Syntax Diagrams appendix.

This manual is also available as the online manual, AFM.

Conventions

When describing NOS/VE command formats, this manual uses the conventions used by the other NOS/VE manuals.

The following conventions are used in this manual.

boldface	Denotes the required parts of a format.
italics	Denotes the optional parts of a format.
blue	Denotes user input within interactive session examples.
UPPERCASE	In formats, denotes the parts of the format that must be entered exactly as shown. In text, names are shown in uppercase.
lowercase	In formats, denotes the parts of the format that the user supplies.
nonproportional typeface	Denotes examples (the nonproportional typeface simulates computer output). User input is indicated by blue print. System output is indicated by black print.
number base	All numbers are decimal unless otherwise indicated.
. . .	In formats, indicates that the preceding items can be repeated.
⋮	In examples, indicates that additional statements would appear at this point, but are not shown.

Vertical bars in the margin indicate changes or additions to the text from the previous revision. An example of a change bar is shown in the margin next to this paragraph.

Submitting Comments

The last page of this manual is a comment sheet. Please tell us about any errors you found in this manual and any problems you had using it.

If the comment sheet in this manual has been used, please send your comments to us at this address:

Control Data Corporation
Technology and Publications Division
P.O. Box 3492
Sunnyvale, California 94088-3492

Include this information with your comments:

The manual title and publication number (for this manual: NOS/VE Advanced File Management Usage, 60486413) and the revision level from the page footer.

Your system's PSR level (if you know it).

Your name, your company's name and address, your work phone number, and whether you want a reply.

Also, if you have access to SOLVER, the Control Data online facility for reporting problems, you can use it to submit comments about this manual. When it prompts you for a product identifier for your report, please specify SM8 for the Sort/Merge documentation, AA8 for the keyed-file utilities documentation, or FM8 for the FMU documentation.

In Case of Trouble

Control Data's CYBER Software Support maintains a hotline to assist you if you have trouble using our products. If you need help beyond that provided in the documentation or find that the product does not perform as described, call us at one of the following numbers and a support analyst will work with you.

From the USA and Canada: (800) 345-9903

From other countries: (612) 851-4131

The preceding numbers are for help on product usage. Address questions about the physical packaging and/or distribution of printed manuals to Literature and Distribution Services at the following address:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

or you can call (612) 292-2101. If you are a Control Data employee, call CONTROLNET® 243-2100 or (612) 292-2100.

Part I: Sort/Merge

Introducing Sort/Merge 1

Introducing Sort/Merge 1

Sort/Merge is the NOS/VE software available for sorting records. Sort/Merge can be executed using a single System Command Language (SCL) command or using procedure calls from within a program written in COBOL, CYBIL, or FORTRAN. For information on using Sort/Merge in a program, refer to the programming language manual (COBOL Usage, CYBIL Keyed-File and Sort/Merge Interfaces, or FORTRAN Language Definition Usage manual). This chapter introduces Sort/Merge use through SCL.

What Sort/Merge Does

The purpose of sorting is to arrange items in order. The purpose of merging is to combine two or more sets of preordered items. Ordered information makes reports more meaningful and suggests critical relationships. Searches for information are faster with ordered lists.

The purpose of Sort/Merge is to arrange records in the sequence you specify. You describe the records you want sorted or merged and information on how Sort/Merge is to order them.

Sort/Merge can:

- Sort or merge in response to a single command entry.
- Use an optional directives file for sort or merge specification.
- Sort or merge records from as many as 100 files with one command.
- Sort character and noncharacter key types.
- Read input records of variable-length (V), ANSI fixed-length (F), or trailing-character-delimited (T) record type.
- Read input records from sequential, indexed-sequential, or direct-access files and write output records to sequential or indexed-sequential files.
- Use mass storage files and magnetic tape files as input and output files.
- Sort using any of twelve predefined collating sequences, thirteen numeric formats, and one or more user-defined collating sequences.

- Sum fields in records that have equivalent key values.
- Use user-defined owncode procedures to insert, substitute, modify, or delete records during the sort or merge.

To start a sort, you enter a SORT command; to start a merge, you enter a MERGE command. Sort/Merge performs the sort or merge based on the parameters that you specify on the SORT or MERGE command. The command parameters are described in chapter 2.

Sort/Merge allows you to specify additional parameters on directives in a directives file. This is provided for two reasons:

- An SCL command parameter can only be specified once on a command. Certain Sort/Merge specifications require that you specify the same parameter more than once.
- The length of an SCL command is limited to 256 characters. A Sort/Merge specification may be longer than that.

Records are sorted or merged on fields of data within each input record; the fields are called sort keys. The data in a sort key field can be 8-bit ASCII character codes, signed or unsigned binary integer, packed decimal, or floating-point numbers. Sort keys are described in detail in later in this chapter.

Depending on the owncode procedures you specify, owncode procedures are executed to process input records, output records, input files, the output file, or records with equal keys. Owncode procedures are described in detail in chapter 3.

Merge capabilities are more restricted than those of a sort. Merge records cannot be supplied by an owncode procedure. Merge input records must be pre-sorted before the files are merged. If the merge specifies summing, the files must also be pre-summed.

Sort/Merge writes the sorted or merged records to a single output file or passes them to an owncode procedure for processing.

Sort Keys

Sort/Merge orders records according to the contents of the sort key fields defined for the sort or merge. The sort key fields are defined by key field definitions on KEY parameters on the SORT or MERGE command. This chapter describes sort keys and the information you supply to define them.

A sort key is a fixed field of data in each input record. This means it must occur in the same position and be the same length in each record. The maximum combined length of all key fields in a record is 1023 bytes. Character data should be left-justified in the field, and numeric data should be right-justified in the field.

Major and Minor Sort Keys

The first key you specify is the most important key and is called the major sort key. This key is sorted or merged first. The keys you specify after the first key are of lesser importance and are called minor sort keys. The minor keys are numbered in the order they are specified. For example, if three sort keys are specified, the first key is the major sort key (key number 1), the next key listed is a minor key (key number 2), and the third key is another minor key (key number 3).

When two or more records have equal major key values, Sort/Merge determines their order by looking at the subsequent minor keys in order (key number 2, key number 3, and so on). Sort/Merge compares the minor keys until either an unequal key is found, or until there are no more keys.

For example, suppose a file of student records is to be sorted on the field of study and then on the student's name. The field of study is specified as the major key so all students with the same field of study are listed together. Because the student name is specified as the minor key, the students with the same field of study are listed in alphabetic order by name.

The file could also be sorted by the class code as the major key and the grade point average as the minor key. If the minor key is sorted in descending order, the sorted list would group the students by class and list the students in each class in order from highest to lowest grade point average.

Describing Sort Keys

If you omit the KEY parameter from the Sort/Merge specification, the default sort key begins with the first byte in the record and extends to the smallest minimum record length value for all input files. If the minimum MINIMUM_RECORD_LENGTH attribute for all input files is 0, Sort/Merge uses 1 as the key length. If the minimum MINIMUM_RECORD_LENGTH attribute for all input files is greater than 1023 bytes, Sort/Merge uses 1023 bytes as the key length.

Otherwise, if you specify one or more KEY parameters, each sort key is defined by a key field definition. Key field definitions include the following information:

- Starting location of the key within the record
- Key length
- Type of data in the key field
- Sort order (ascending or descending)

Sort/Merge allows key fields to overlap other key fields, except for the following:

- Key fields that are ordered by collating sequences defined with the alter option
- Key fields that overlap sum fields

Key Field Definition

The key field definitions are specified on the KEY parameter on the SORT or MERGE command or in a KEY parameter on a directive in the directives file. A key field definition is a value set of up to four values. Only the first value, specifying the key position, is required.

If a key field definition specifies more than one value, it must be enclosed in parentheses. Values within a value set are separated by a comma or a space.

A key field definition has one of the following formats:

```
(position..position, key_type, sort_order)
(position, length, key_type, sort_order)
```

Byte and bit positions in the record are numbered from the left beginning with 1. Sort/Merge interprets the key field position and length specification as bytes unless the key type is INTEGER_BITS or BINARY_BITS.

Table 1-1 lists the maximum key field sizes.

Table 1-1. Maximum Key Field Sizes

Key Type	Maximum Size (in bytes)	Key Type	Maximum Size (in bytes)
Character	1023	BINARY	8
NUMERIC_FS	1023	BINARY_BITS	8184 (bits)
NUMERIC_LO	38	INTEGER	8
NUMERIC_LS	38	INTEGER_BITS	8184 (bits)
NUMERIC_NS	38	PACKED	19
NUMERIC_TO	38	PACKED_NS	19
NUMERIC_TS	38	REAL	8 or 16

Key Field Definition Using a Range

If the first value in the key field definition is a range, it defines the position and length of the key field. The range specifies as its upper and lower bounds the first and last positions in the key field. For example, the range 1..3 defines a key field from position 1 through position 3.

The lower bound of the range (its second value) is optional. When omitted, the lower bound is assumed to be the same as the upper bound so the length of the key field is 1.

Key Field Definition Without a Range

If the first value in the key field definition is not a range, the first two values in the definition are integers that define the position and length of the key field. The first value specifies the first byte or bit position in the field. The second value specifies the number of bytes or bits in the field.

The first value, specifying the key position, is required. If you omit the second value, the key length is assumed to be 1.

Key Type

After specifying the key field position and length, the key field definition specifies the type of data in the key field. It can be the name of a collating sequence or the name of a numeric data format.

By default, the key field is assumed to contain ASCII character data to be sorted according to the default ASCII collating sequence.

The key type specified depends on the contents of the sort key field and on whether the key is to be sorted in numerical order or in collating sequence order:

- If the data is ASCII character codes, but it represents a number to be sorted by numerical value, the key type should specify a numeric data format.
- If the ASCII character codes are to be sorted byte-by-byte according to a collating sequence, the key type should specify a collating sequence.
- If the data is an arithmetic representation of a number (binary, integer, floating-point, or packed-decimal), the key type should specify the corresponding numeric format.

If a key field contains data that is not meaningful for the key type you specify (such as an alphabetic character in a numeric character field), Sort/Merge determines that the field contains invalid data and so cannot be sorted. If an exception records file has been specified for the sort or merge, the record is removed from the sort or merge and written to exception records file. Otherwise, the record remains in the sort or merge, but its place in the sort order is undefined.

Collating Sequences

A collating sequence determines the precedence given to each character in relation to the other characters. Character data must be in ASCII character codes.

If you do not specify a key type, the default ASCII collating sequence (ASCII) is used. ASCII is the fastest predefined character collating sequence.

NOS/VE has twelve predefined collating sequences. Sort/Merge allows you to use six of these collating sequences without explicitly loading the collation table. The six are:

Key Type	NOS/VE Predefined Collation Table
ASCII	The default ASCII collating sequence
ASCII6	OSV$ASCII6_FOLDED
COBOL6	OSV$COBOL6_FOLDED
DISPLAY	OSV$DISPLAY64_FOLDED
EBCDIC	OSV$EBCDIC
EBCDIC6	OSV$EBCDIC6_FOLDED

To use the other predefined NOS/VE collating sequences you must explicitly load the collating sequence by specifying a LOAD_COLLATING_TABLE parameter. The predefined collating sequences are listed in appendix E.

You can also create your own collating sequence using the COLLATING_SEQUENCE_x parameters described in chapter 2.

Numeric Data Formats

The available numeric data formats are listed in table 1-2.

For Better Performance

Of the numeric data formats, the most efficient key types are INTEGER, BINARY, and REAL.

Table 1-2. Numeric Data Formats

Name	Data Type	Sign	Comments
BINARY	Binary integer	None	The field must start and end on character boundaries.
BINARY_BITS	Binary integer	None	The field need not start or end on character boundaries.
INTEGER	Two's complement binary integer	Positive if leftmost bit is 0; negative if leftmost bit is 1	The field must start and end on character boundaries.
INTEGER_ BITS	Two's complement binary integer	Positive if leftmost bit is 0; negative if leftmost bit is 1	The field does not start or end on character boundaries.

(Continued)

Table 1-2. Numeric Data Formats *(Continued)*

Name	Data Type	Sign	Comments
NUMERIC_FS	Leading blanks, numeric characters	- sign for negative values; a + character is not allowed	The field contains leading blanks (leading zeros must be converted to blanks before calling Sort/Merge); if the value is negative, the rightmost leading blank must be converted to a minus sign. If the field contains no leading blanks or does not begin with a negative sign, the value must be positive. This format is equivalent to the FORTRAN I format, or the COBOL picture clause for zero suppressed editing of numeric item.
NUMERIC_LO	Numeric characters	Leading overpunch	All characters are decimal digits except the leading character, which indicates a sign by an overpunch. All forms of zero are ordered equally.

(Continued)

Table 1-2. Numeric Data Formats *(Continued)*

Name	Data Type	Sign	Comments
NUMERIC_LS	Numeric characters	Leading separate	All characters are decimal digits except the leading character, which is a negative or positive sign. Specifying a field that is not at least two characters in length causes a fatal error. All forms of zero are ordered equally.
NUMERIC_NS	Numeric characters	None	All characters are decimal digits.
NUMERIC_TO	Numeric characters	Trailing overpunch	All characters are decimal digits except the trailing character, which indicates a sign by an overpunch. All forms of zero ordered equally.
NUMERIC_TS	Numeric characters	Trailing separate	All characters are decimal digits except the trailing character, which is a negative or positive sign. Specifying a field that is not at least two characters in length causes a fatal error. All forms of zero ordered equally.

(Continued)

Table 1-2. Numeric Data Formats *(Continued)*

Name	Data Type	Sign	Comments
PACKED	Packed decimal	Signed	Data is ordered according to numeric value.
PACKED_NS	Unsigned packed decimal	Unsigned	PACKED_NS is the same as COBOL COMPUTATIONAL-3 with no sign.
REAL	Normalized floating-point number, either single-precision (8 bytes) or double-precision (16 bytes)	Signed	All forms of zero are ordered equally. The order of indefinite values is undefined. Infinite values are ordered as if their value were infinity (can be signed infinity).

Floating Sign

The NUMERIC_FS format contains a floating sign if the value is negative. This means that the character preceding the numeric characters must be a minus (-) character. All leading characters must be blanks. Positive values in this format are not signed. The following examples are valid floating sign formats:

```
    - 1
      1
    - 0
      0
- 1 2 3
1 2 3 4
```

The following examples are invalid floating sign formats:

```
  0 1   Leading zero not allowed
- 0 1   Leading zero not allowed
+ 1 2 3 Positive sign not allowed
        All blank field not allowed
```

Sort/Merge issues diagnostic messages for invalid floating sign formats.

Overpunch Sign

A negative sign overpunch is equivalent to overstriking a digit with a -, which is a punch in row 11 of a punched card. A positive sign overpunch is equivalent to overstriking a digit with a +, which is a punch in row 12 of a punched card.

When a signed overpunch digit is received as input, the digit is punched as indicated in the second column of table 1-3. When a signed overpunch digit is entered from a terminal or displayed as output, the digit appears as indicated in the third column of table 1-3. The hexadecimal value is in the fourth column.

Sort Order

The optional fourth value in the value set specifies the sort order. Sort order is either ascending or descending as indicated by the keyword value A or D, respectively. If you specify neither, the sort order is assumed to be ascending.

When sorting a numeric key in ascending order, Sort/Merge sorts the key values in numeric order from least to greatest. When sorting a numeric key in descending order, Sort/Merge sorts the key values in numeric order from greatest to least.

A character key is sorted according to the collating sequence you specify for the key. For ascending order, the key values are sorted in the order given by the collating sequence, For descending order, the key values are sorted in reverse order from the collating sequence.

Table 1-3. Sign Overpunch Representation

Sign and Digit	Input Punch	Input/Output Representation	Hexadecimal Value
+0	0	0	30
+1	1	1	31
+2	2	2	32
+3	3	3	33
+4	4	4	34
+5	5	5	35
+6	6	6	36
+7	7	7	37
+8	8	8	38
+9	9	9	39
+0	12-0	{	7B
+1	12-1	A	41
+2	12-2	B	42
+3	12-3	C	43
+4	12-4	D	44
+5	12-5	E	45
+6	12-6	F	46
+7	12-7	G	47
+8	12-8	H	48
+9	12-9	I	49
-0	11-0	}	7D
-1	11-1	J	4A
-2	11-2	K	4B
-3	11-3	L	4C
-4	11-4	M	4D
-5	11-5	N	4E
-6	11-6	O	4F
-7	11-7	P	50
-8	11-8	Q	51
-9	11-9	R	52
+0	12-8-4	<	3C
+0	12	&	26
-0	12-8-7	!	21
-0	11	-	2D

Specifying the Record Length

Sort/Merge can sort records up to 65,535 bytes long. Sort/Merge determines the maximum and minimum record lengths for a file by its MAXIMUM RECORD_LENGTH and MINIMUM_RECORD_LENGTH file attributes.

The record length attributes are set when the file is created. You can specify the record for a new file with a SET_FILE_ATTRIBUTE command. If you specify an already existing file on the SET_FILE_ATTRIBUTE command, the command is ignored. For details on the SET_FILE_ATTRIBUTE command, refer to the NOS/VE System Usage manual.

The default sort key begins with the first byte in the record and extends to the smallest minimum record length value for all input files. If the minimum MINIMUM_RECORD_LENGTH attribute for all input files is 0, Sort/Merge uses 1 as the key length. If the minimum MINIMUM_RECORD_LENGTH attribute for all input files is greater than 1023 bytes, Sort/Merge uses 1023 bytes as the key length.

Sort performance is best when the maximum record length attribute value is equal to the longest record to be sorted. Sort/Merge can sort records up to 65,535 bytes long.

If the SORT command specifies an owncode 1 procedure to supply input records and an owncode 3 procedure to perform output processing and omits the FROM and TO parameters, the command must specify the record length using either the OWNCODE_FIXED_LENGTH or OWNCODE_MAXIMUM_RECORD_LENGTH parameter.

Short Records

A short record is a record that does not contain all key and sum fields defined for the sort or merge. Sort/Merge determines that a record is short when it reads the record from the input source. Therefore, missing or partial key and sum fields are detected even if Sort/Merge does not use their contents.

NOTE

Records can become short when the system strips off trailing blanks from variable-length (V) records. For example, when a variable-length record containing all spaces is displayed by the SCL command DISPLAY_FILE, the spaces are stripped from the record, leaving a zero-length record.

When Sort/Merge finds that a key or sum field is entirely beyond the end of the record, it uses a default value for the field. For character keys, the default value is all spaces. For numeric keys and sum fields, the default value is zero in the appropriate format.

The default value does not actually become part of the record data. Sort/Merge uses the default value only when using the key value or sum field value. It does not pass the default value to an owncode procedure or store the default value in the output record.

Sort/Merge processing differs when the field is only partially beyond the end of the record. If the partial field is a character key field, Sort/Merge pads it with spaces, but if the partial field is a numeric key field or sum field, Sort/Merge processes it as an exception.

Exception processing for partial sum fields is described in detail under the SUM topic in chapter 2. Exception processing for partial numeric key fields is as follows:

1. The record is written to the exception records file if one is specified for the sort or merge.
2. If an exception records file exists, the record is removed from the sort or merge; otherwise, its order is left undefined.
3. The count of partial numeric key fields is incremented. A warning error message gives the count at the end of the sort or merge.

Zero-Length Records

A zero-length record is a record that contains no data and so its record length is 0. The processing of zero-length records read from input files depends on the ZERO_LENGTH_RECORDS parameter specification.

By default, Sort/Merge deletes all zero-length records from the sort or merge. This is the DELETE option.

However, instead of the DELETE option, ZERO_LENGTH_RECORDS can specify one of the following options for zero-length records:

PAD Assigns default values to key fields and sum fields in zero-length records (as it would short records) and keeps the zero-length records in the sort or merge.

LAST Writes zero-length records at the end of the output file.

Zero-length records are never written to the exception records file if the DELETE option is selected.

Zero-length records are written to the exception records file if the PAD option is selected and either of the following situations exist:

- If merge order verification is requested and the input files contain zero-length records that are not pre-sorted on the merge keys.
- If the system procedures that writes the record (AMP$PUT_NEXT) detects an error while writing a zero-length record. (In general, attempts to write zero-length records to an indexed-sequential file cause errors.)

If OMIT_DUPLICATES=YES and ZERO_LENGTH_RECORDS=PAD are both specified, only one zero-length record is included in the sort or merge.

Zero-length records are passed to owncode procedures only if the PAD option is selected. When passing a zero-length record, Sort/Merge passes an empty array of the maximum record length and a record length of zero.

The count kept in the result array for the sort or merge may differ depending on the ZERO_LENGTH_RECORDS specification:

Element 2, number of records read:

Zero-length records are always included in the count.

Element 6, number of records sorted or merged:

Zero-length records are included only if PAD is selected.

Elements 13, 14, and 15, number of records written, the minimum record length, and the average record length:

Zero-length records are included in these values only if PAD or LAST is selected.

Element 17, the number of zero-length records deleted from the sort or merge:

This count is kept only if DELETE is selected.

Invalid Records

Sort/Merge checks that the data in all key fields is valid. It determines whether the data in sum fields is valid only when it attempts to use the data. It does not validate the data in any other record fields.

If an exception records file is specified, Sort/Merge copies each invalid record to the exception records file. It then removes the invalid record from the sort or merge. Therefore, if all input records are invalid and an exception records file is specified, no records are written to the output file.

If an exception records file is not specified, records with invalid key values or sum values are not deleted from the sort or merge. The order of records with invalid key fields is undefined. The contents of sum fields with invalid data is also undefined.

Write Errors

Sort/Merge also considers a record to be invalid if an error is returned by an attempt to write the record. Sort/Merge writes records to the output file using the system procedure AMP$PUT_NEXT. If AMP$PUT_NEXT returns an error for a record, Sort/Merge writes the records to the exception records file (if one is specified) and deletes it from the sort or merge.

AMP$PUT_NEXT may return errors (such as duplicate primary-key value) when writing to an indexed-sequential file. The invalid record is written to the exception records file (if one is specified) and deleted from the sort or merge.

Performance Considerations

To improve Sort/Merge performance, consider the following:

- Do not use owncode procedures except when necessary.
- Ensure that all key fields and sum fields are within the minimum record length for all input records. Additional processing is required for short records.
- If possible, use a fixed record length instead of a variable record length.
- Of the numeric data formats, the most efficient key types are INTEGER, BINARY, and REAL.
- Sort/Merge can read and write files faster if the files use the default attributes, as follows:
  - Sequential file organization
  - F or V record type
  - System-specified blocking
  - No error-exit procedure
  - No file access procedure (FAP)
  - The padding character is space

Limiting Memory Usage

By default, Sort/Merge limits the memory assigned to its sorting array to 262,144 (256K) bytes.

You can change the Sort/Merge memory limit by defining an SCL integer variable named SMV$MEMORY_USAGE_LIMIT. The integer you assign to the variable is used as the memory usage limit for subsequent sorts within the scope of the variable. For example, the following command creates the SMV$MEMORY_USAGE_LIMIT variable and assigns it the value 64.

```
create_variable, smv$memory_usage_limit, kind=integer, ..
value=64, scope=job)
```

The integer that you specify is multiplied by 1024 (1K) to determine the limit in bytes. The minimum limit is 64; if you specify an integer less than 64, Sort/Merge uses 64. The maximum limit is 16,383; if you specify an integer greater than 16,383, Sort/Merge uses 16,383. A warning error is issued when you specify a value outside the range of 64 through 16,383.

The SMV$MEMORY_USAGE_LIMIT value is not used to limit memory usage for merges; it is used only for sorts (including the internal merge that is part of a sort).

Setting the Page_Aging_Interval

The page_aging_interval is the job attribute that controls how quickly pages are aged from the working set of a task. If you increase the memory usage limit for your sorts, you should also increase your page_aging_interval value.

The optimum page_aging_interval depends on the CYBER 180 model you use. A smaller value is appropriate for a faster models. For example, when the default memory usage limit of 256 pages is used, the optimum page_aging_interval for a CYBER 180/830 is about 500,000 microseconds, while for a CYBER 180/860, the optimum value is about 100,000 microseconds.

To see your current page_aging_interval attribute value, enter the following SCL command:

```
display_job_attribute, display_option=page_aging_interval
```

To change your page_aging_interval value, use the CHANGE_JOB_ATTRIBUTE command. For example, the following command changes the page_aging_interval to 500,000 microseconds:

```
change_job_attribute, page_aging_interval=500000
```

The SCL Commands SORT and MERGE 2

The SCL Commands SORT and MERGE 2

This chapter describes the use of Sort/Merge via the System Command Language (SCL) commands SORT and MERGE. It first describes the command format and then provides detailed individual descriptions of each parameter.

Specifying Parameters by Position

As on any SCL command, a Sort/Merge parameter can be specified without its parameter name. In this case, the parameter value is assigned to a parameter by its position in the parameter sequence. Table 2-1 lists the positional order of the Sort/Merge parameters.

For example, both of the following SORT commands specify parameter values for the FROM, TO, and EXCEPTION parameters (positions 1, 2, and 11).

```
sort, $user.input1, $user.output2,,,,,,,,,$user.exception_file
sort, from=$user.input1, to=$user.output2, ..
erf=$user.exception_file
```

As you can see by the example, it is more difficult to see which parameters have values specified when the values are specified by position. Use of the parameter names is recommended.

Table 2-1. Parameter Positional Order

Position	Parameter Name	Position	Parameter Name
1	FROM	16	OWNCODE_PROCEDURE_1
2	TO	17	OWNCODE_PROCEDURE_2
3	KEY	18	OWNCODE_PROCEDURE_3
4	DIRECTIVES_FILE	19	OWNCODE_PROCEDURE_4
5	LIST	20	OWNCODE_PROCEDURE_5
6	LIST_OPTIONS	21	RETAIN_ORIGINAL_ORDER
7	ERROR	22	COLLATING_SEQUENCE_NAME
8	ERROR_LEVEL	23	COLLATING_SEQUENCE_STEP
9	Reserved	24	COLLATING_SEQUENCE_REMAINDER
10	ESTIMATED_NUMBER_RECORDS	25	COLLATING_SEQUENCE_ALTER
11	EXCEPTION_RECORDS_FILE	26	STATUS
12	C170_COMPATIBLE	27	SUM
13	OMIT_DUPLICATES	28	ZERO_LENGTH_RECORDS
14	OWNCODE_FIXED_LENGTH	29	VERIFY_MERGE_INPUT_ORDER
15	OWNCODE_MAXIMUM_RECORD_LENGTH	30	LOAD_COLLATING_TABLE
		31	RESULT_ARRAY

Specifying Parameters in Directive Files

You can use directive files to repeat Sort/Merge parameters. A parameter can appear only once on an SCL command. However, if so indicated in the individual parameter description, the parameter can appear on the command and/or on directives in a directive file.

Also, an SCL command can be no longer than 256 characters long. When your Sort/Merge specifications are longer than 256 characters, you can continue the parameter specifications in directives files.

A sort or merge can use up to 100 directive files. The first list of directive files is specified on the DIRECTIVES_FILE parameter on the SORT or MERGE command. Each directive can also specify a list of directive files on a DIRECTIVES_FILE parameter. Figure 2-1 illustrates the order in which directive files are read.

A Sort/Merge directives file contains one or more directives. It can also contain SCL commands and comments. (Each SCL comment begins with a quote ["] character.)

Sort/Merge directives have the same format as a SORT or MERGE command and are processed according to SCL command conventions. Like SCL commands, each directive:

- Begins with the word SORT or MERGE followed by a comma or space.
- Can specify each parameter only once.
- Can be up to 256 characters in length.
- Can continue over more than one line. A line to be continued ends with two or more periods. The continuation periods are not included in the directive length.

NOTE

Because a range is also specified using two periods (for example, 1..3), do not split a command or directive within a range.

For example, the directive SORT,OLD,NEW,KEY=(1..10),DIRECTIVES_FILE=FILE1 can be written as follows:

```
SORT,OLD,NEW,..
KEY=(1..10), DIRECTIVES_FILE=..
FILE1
```

In general, Sort/Merge specifications can use uppercase or lowercase letters. However, an exception exists for owncode procedure names. Unless you specify C170_COMPATIBLE=YES, all owncode procedure names must be specified using uppercase letters only. (This is because entry point names are stored using only uppercase letters and no conversion is performed unless requested.)

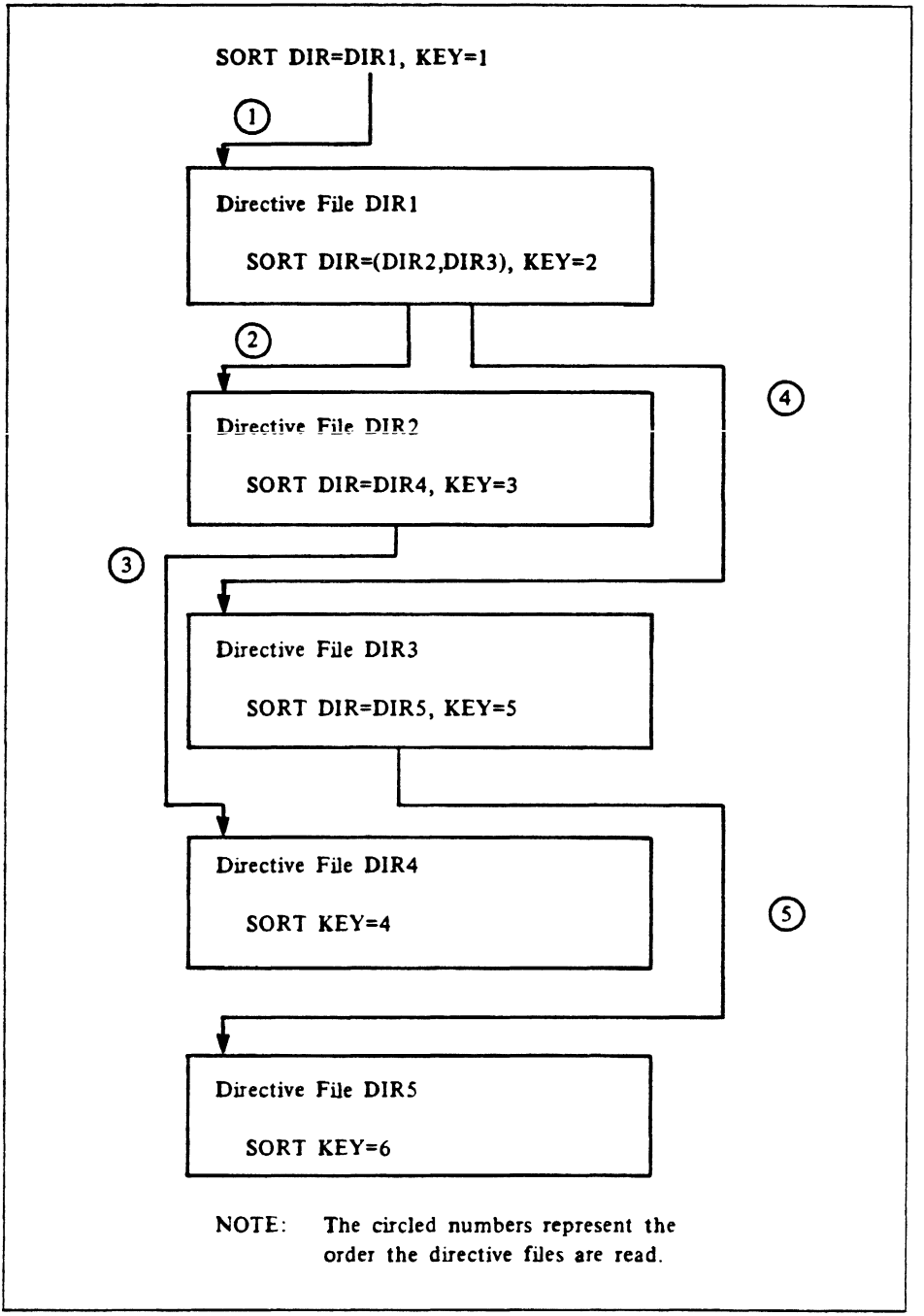


Figure 2-1. Directive File Order

The Sort/Merge Parameters

This section discusses each Sort/Merge parameter in detail. The parameter descriptions are presented in alphabetical order. The parameter name abbreviations appear in parentheses after the parameter name.

You can enter the Sort/Merge parameter values using uppercase, lowercase, or a combination of uppercase and lowercase. The one exception is owncode procedure names, which must be specified using all uppercase unless you specify C170_COMPATIBLE=YES.

C170_COMPATIBLE (CC)

Purpose Specifies whether lowercase letters in owncode procedure names are to be converted to uppercase letters. This is required for loading of the owncode procedures.

Default:

If you omit the C170_COMPATIBLE parameter, the default is OFF and the owncode procedure names are not converted. Therefore, the names must be specified using uppercase letters.

Format *C170_COMPATIBLE = boolean*

An SCL boolean is a logical true or false value specified by the keyword YES, TRUE, or ON for true or NO, FALSE, or OFF for false. A true specification indicates that Sort/Merge converts owncode procedure names to uppercase letters, if necessary. A false specification indicates Sort/Merge does not convert lowercase letters; names must be specified using uppercase letters.

Remarks

- When Sort/Merge attempts to load an owncode procedure, it passes the procedure name as you have specified it on the OWNCODE_PROCEDURE_n parameter. If you specify the name with lowercase letters, Sort/Merge passes the lowercase letters unless the C170_COMPATIBLE parameter requests conversion.
- The system stores entry point names using uppercase letters only. Therefore, if the loader is given a procedure name containing lowercase letters, it cannot find that name in the program library list and so it cannot load the requested procedure.

COLLATING_SEQUENCE_x (CSx or SEQx)

The COLLATING_SEQUENCE_x parameters allow you to define your own collating sequence. (You can also load your own collating sequences using LOAD_COLLATING_TABLE parameters. The total number of user-defined collating sequences used by a sort or merge cannot exceed 100.)

A collating sequence specifies the sort or merge order for character data. It defines the collating position assigned to each of the 256 ASCII characters. Any characters not explicitly assigned a collating position are assigned to the last position in the collating sequence.

A collating sequence consists of a series of value steps; each value step in the sequence is assigned a collating position from lowest to highest. Each value step contains at least one ASCII character. When a value step contains more than one character, all characters in the step have the same collating weight and are collated equally.

For example, suppose a collating sequence has 27 value steps. The 26 letters of the alphabet are each assigned a value step in standard alphabetical order (A through Z). The rest of the ASCII character set is assigned to the 27th value step. Using this collating sequence, a sorted sequence would have the letters sorted first from A through Z followed by all non-alphabetic characters collated equally. (The order of the non-alphabetic characters is random unless other processing is specified by equivalent key values.)

Each collating sequence definition begins with a COLLATING_SEQUENCE_NAME parameter and continues until Sort/Merge reads a parameter other than COLLATING_SEQUENCE_STEP, COLLATING_SEQUENCE_REMAINDER, or COLLATING_SEQUENCE_ALTER.

The default ASCII collating sequence assigns one character to each value step. The value steps are ordered as the characters are ordered in the ASCII character set listing in appendix C.

COLLATING_SEQUENCE_NAME (CSN or SEQN)

Purpose Marks the start of a collating sequence definition and specifies the name of the collating sequence.

Format *COLLATING_SEQUENCE_NAME=name*

Default:

None. The COLLATING_SEQUENCE_NAME parameter is required to begin a Sort/Merge collating sequence definition. (Collating sequence definitions are optional.)

Remarks

- The COLLATING_SEQUENCE_NAME parameter can be specified once on the command and once on each directive. It can be specified only once per collating sequence definition.
- The specified collating sequence name cannot be the name of a predefined collating sequence or a collating sequence already defined for the sort or merge.
- The collating sequence name is specified as the key type for the key field to be sorted or merged according to the collating sequence. For example, the following parameter begins a collating sequence definition and names the collating sequence MYSEQUENCE:

```
COLLATING_SEQUENCE_NAME=mysequence
```

  The following key definition indicates that the key contains character data to be sorted according to the MYSEQUENCE collating sequence:

```
KEY=((1..10,mysequence))
```

- The SEQN abbreviation is provided for CYBER 170 SORT5 compatibility; its continued use is not recommended.

COLLATING_SEQUENCE_STEP (CSS or SEQS)

Purpose Defines one or more value steps within the collating sequence.

Format *COLLATING_SEQUENCE_STEP=list of value step definitions*

Default:

None. At least one COLLATING_SEQUENCE_STEP parameter must be specified in a Sort/Merge collating sequence definition.

Remarks

- The SEQS parameter can be specified once on the command and once on each directive. Multiple COLLATING_SEQUENCE_STEP parameters can be specified in each collating sequence definition.
- The position of the defined value steps within the sequence is indicated by the position of the COLLATING_SEQUENCE_STEP parameter within the collating sequence definition. For example, the second COLLATING_SEQUENCE_STEP parameter in the collating sequence definition defines the second value step in the collating sequence.
- You can specify a single character either by the ASCII graphic character enclosed in apostrophes or by the character ordinal in the SCL function $CHAR. The character ordinal is the position of the character within the ASCII character set as shown in appendix C. For example, the character A can be specified as 'A' or $CHAR(65).

  The apostrophe character can be specified as four apostrophes ('''') or $CHAR(39).
- The SEQS abbreviation is provided for CYBER 170 SORT5 compatibility; its continued use is not recommended.

Single Value Step of a Single Character

A COLLATING_SEQUENCE_STEP parameter can define a single value step containing one character. For example, the following two parameters each specify a value step containing a single letter (the letters A and B).

```
COLLATING_SEQUENCE_STEP='A'
COLLATING_SEQUENCE_STEP=($CHAR(66))
```

Single Value Step of Several Characters

A COLLATING_SEQUENCE_STEP parameter can define a single value step containing several characters. To do so, it specifies a list beginning with a single character followed by one or more single characters or ranges of characters. The list is enclosed in parentheses.

For example, the following parameter specifies a single value step containing the blank character and the digits 0, 1, 2, and 3, which collate equally.

```
COLLATING_SEQUENCE_STEP=(' ', '0', '1', '2', '3')
```

Another way of specifying the same value step is shown below. The range '0'..'3' specifies the range of digits from 0 through 3.

```
COLLATING_SEQUENCE_STEP=(' ', '0'..'3')
```

A fatal error is issued if a COLLATING_SEQUENCE_STEP specification begins with a range followed by one or more single character specifications or ranges of a different size.

Several Value Steps of One Character

A COLLATING_SEQUENCE_STEP parameter can define several value steps, each containing one character. To do so, it specifies a single range of characters. The range defines a sequence of value steps, one for each character in the range.

For example, to specify a sequence of four value steps each containing one character, for the characters 0, 1, 2, and 3, you can use either the single value step definition on the left or the four value step definitions on the right:

```
COLLATING_SEQUENCE_STEP=('0'..'3')  COLLATING_SEQUENCE_STEP=('0')
                                    COLLATING_SEQUENCE_STEP=('1')
                                    COLLATING_SEQUENCE_STEP=('2')
                                    COLLATING_SEQUENCE_STEP=('3')
```

Several Steps of Several Characters

A COLLATING_SEQUENCE_STEP parameter can define several value steps, each containing more than one character. Each value step is assigned one character from each of the ranges specified on the parameter. The character has the same position within each of the ranges. Therefore, all specified ranges must be the same size.

For example, suppose the collating sequence is to assign equal value to uppercase and lowercase letters. This requires definition of 26 value steps each containing an uppercase letter and a lowercase letter. The first value step should contain both A and a, the second value step B and b, and so forth. You can define the 26 value steps for the alphabetic characters using this parameter:

```
COLLATING_SEQUENCE_STEP('a'..'z', 'A'..'Z')
```

If the COLLATING_SEQUENCE_STEP specification begins with a range, all subsequent ranges must be the same size; otherwise, a fatal error is issued.

COLLATING_SEQUENCE_REMAINDER (CSR or SEQR)

Purpose Defines a special value step. This special step contains all characters not specified by other COLLATING_SEQUENCE_STEP parameters within the collating sequence definition.

Default:

When you omit the COLLATING_SEQUENCE_REMAINDER parameter, Sort/Merge assumes a true specification and creates a value step containing all unspecified characters as the last value step in the collating sequence.

Format *COLLATING_SEQUENCE_REMAINDER = boolean*

An SCL boolean is a logical true or false value. A true specification (TRUE) indicates that the special value step is used in the collating sequence; a false specification (FALSE) indicates that the special value step is not used.

Remarks

- The SEQR abbreviation is provided for CYBER 170 SORT5 compatibility; its continued use is not recommended.
- A directives file can contain more than one collating sequence definition. The COLLATING_SEQUENCE_REMAINDER parameter can appear only once in a collating sequence definition. It can appear anywhere in the definition after the COLLATING_SEQUENCE_NAME parameter.

Examples

The following directive sequence defines a collating sequence that uses the special value step. In this case, the special value step contains all nondigits and nonletters (such as periods, commas, and slashes):

```
SORT, COLLATING_SEQUENCE_STEP=('0'..'9')
SORT, COLLATING_SEQUENCE_REMAINDER=YES
SORT, COLLATING_SEQUENCE_STEP=('A'..'Z')
```

This sequence defines value steps in the following order: digits in numeric order, nondigits and nonletters, and letters in alphabetic order. The nondigits and nonletters have equal collating positions.

COLLATING_SEQUENCE_ALTER (CSA or SEQA)

Purpose

Used with the COLLATING_SEQUENCE_STEP parameter to specify whether characters are altered in the output. If characters are altered, all characters within a value step specified on a COLLATING_SEQUENCE_STEP parameter are output as the first character in the value step.

Default:

If this parameter is omitted, the characters in the value step are not altered.

Format

COLLATING_SEQUENCE_ALTER = boolean

An SCL boolean is a logical true or false value specified by the keyword YES, TRUE, or ON for true or NO, FALSE, or OFF for false. A true specification indicates that characters are altered; a false specification indicates that characters are not altered.

Remarks A directives file can contain more than one collating sequence definition. The COLLATING_SEQUENCE_ALTER parameter can appear only once in a collating sequence definition. It can appear anywhere in the definition after the COLLATING_SEQUENCE_NAME parameter.

Examples The following sequence alters all asterisks and ampersands to slashes in the output:

```
SORT, COLLATING_SEQUENCE_STEP='/', '*', '&')
SORT, COLLATING_SEQUENCE_ALTER=YES
```

Storing a Collating Sequence Definition in a File

Collating sequences are usually defined in directive files. This is because almost all collating sequence definitions require more than one COLLATING_SEQUENCE_STEP parameter, but only one COLLATING_SEQUENCE_STEP parameter can appear on a SORT or MERGE command.

It is also convenient to store the directives defining a collating sequence in a file so that the collating sequence can be reused. A collating sequence definition in a directive file can then be used by any sort or merge that specifies the file.

DIRECTIVES_FILE (DF or DIR or DIRECTIVES)

Purpose Specifies one or more directive files from which sort or merge directives are read.

Format *DIRECTIVES_FILE=list of file*

Default:

If you omit the DIRECTIVES_FILE parameter, no parameters are read from a directive file; the sort or merge is completely specified on the SORT or the MERGE command.

Remarks

- Use of directives files is described at the beginning of this chapter under the heading Specifying Parameters in Directives Files. An example of directives file use is given in chapter 11.
- Parameters are read from directives only after all command parameters have been read. When more than one directive file is specified, Sort/Merge reads the directive files in the following order:

  1. The first directive file
  2. All subsequent directive files referenced by the first directive file or referenced by the subsequent files
  3. Directive file(s) named after the first directive file and the subsequent files, as indicated above

- A directive file name referenced without a file path is assumed to be in the working catalog unless the file name is for a standard system file. Standard system files, such as $INPUT or $OUTPUT, are assumed to be in the $LOCAL catalog.
- If Sort/Merge cannot access a specified file, it issues a warning message.
- The DIRECTIVES and DIR abbreviations are provided for CYBER 170 SORT5 compatibility; their continued use is not recommended.

ERROR (E)

Purpose Specifies the file to which diagnostic messages are written.

Format *ERROR=file*

Default:

If you omit the parameter, diagnostic messages are written to file $ERRORS.

Remarks

- Sort/Merge writes the error file only if it detects errors of at least the severity specified by the ERROR_LEVEL parameter.
- The error file is not rewound before or after the Sort/Merge operation unless repositioning is requested by the file position indicator ($BOI) on the file reference.
- If you specify ERROR=$NULL, diagnostic messages are not written.
- If you specify the same file for the listing file and for the error file, each error diagnostic message is written only once to the file. Otherwise, each message is written twice, once to the listing file and once to the error file.
- An error file name referenced without a file path is assumed to be in the working catalog unless the file name is for a standard system file. Standard system files, such as $INPUT or $OUTPUT, are assumed to be in the $LOCAL catalog.

ERROR_LEVEL (EL)

Purpose Specifies the minimum error severity of the diagnostic messages written to the error file.

Default:

If you omit the ERROR_LEVEL parameter, only warning, fatal, and catastrophic messages are written to the error file.

Format *ERROR_LEVEL=keyword*

The valid keyword values are as follows:

INFORMATIONAL (I)	Report informational, warning, fatal, and catastrophic errors
TRIVIAL (T)	Same as informational (This is a nonstandard value and its use is not recommended)
WARNING (W)	Report warning, fatal, and catastrophic errors only
FATAL (F)	Report fatal and catastrophic errors only
CATASTROPHIC (C)	Report catastrophic errors only
NONE	Report no errors

Remarks

- A Sort/Merge error can be one of the following error severities:

Informational	An informational error results from a usage that is syntactically correct but questionable. An informational message is issued.
Warning	A warning error results when Sort/Merge finds an error but recovers by making assumptions about your attempt.

Fatal	A fatal error results when Sort/Merge cannot resolve an error. Sort/Merge treats error severities ERROR and FATAL as fatal errors.
Catastrophic	A catastrophic error causes immediate Sort/Merge termination.

ESTIMATED_NUMBER_RECORDS (ENR)

Purpose Although you can specify a value on the ESTIMATED_NUMBER_RECORDS parameter, NOS/VE Sort/Merge does not use the value. The parameter exists to provide compatibility with the CYBER 170 SORT5 product.

Format *ESTIMATED_NUMBER_RECORDS=range*

An SCL range is specified as two integer values separated by two periods (..). The minimum lowerbound on an ESTIMATED_NUMBER_RECORDS range is 1; the maximum upperbound is 16777215.

EXCEPTION_RECORDS_FILE (ERF)

Purpose Specifies the file to which invalid records are written.

Format *EXCEPTION_RECORDS_FILE=file*

Default:

If you omit the parameter, invalid records are written with the valid records on the output file. The order of the records with invalid key fields is undefined; the contents of invalid sum fields is also undefined.

Remarks

- The file specified as the exception records file cannot also be specified as an output file on the TO parameter.
- If you specify EXCEPTION_RECORDS_FILE=$NULL, Sort/Merge deletes all records that would be written to the exception records file.
- An exception records file name referenced without a file path is assumed to be in the working catalog unless the file name is for a standard system file. Standard system files, such as $INPUT or $OUTPUT, are assumed to be in the $LOCAL catalog.
- The exception records file cannot be a keyed file.
- The records written to the exception file include the following:
  - Records containing invalid key data.
  - Records containing invalid sum data if summing is attempted. (For more information, see the SUM parameter description.)
  - Records that caused an arithmetic overflow or underflow condition when their sum fields were summed.
  - Short records containing a partial numeric key field or a partial sum field.
  - Out-of-order merge input records if the VERIFY=YES is specified.

  - Records for which an error was returned when the system procedure AMP$PUT_NEXT attempted to write the record to the output file.

- For additional information on the processing of short and invalid records, see Short Records and Invalid Records in chapter 3.

- Records written to the exception records file are deleted from the sort or merge. A summary of records written to the exception records file is printed in the error and list files. If the DE option is specified on the LIST_OPTIONS parameter, detailed exception information is written to the error and list files.

FROM (F)

Purpose Specifies one or more input files from which input records are read.

NOTE

Merge input files must be pre-sorted. Files to be summed by a merge must be pre-sorted and pre-summed.

Format *FROM=(list of files)*

Default:

If you omit the FROM parameter but specify the OWNCODE_PROCEDURE_1 parameter, Sort/Merge assumes that input records are provided by the owncode 1 procedure; it does not require an input file in this case.

If you omit the FROM parameter and the OWNCODE_PROCEDURE_1 parameter, Sort/Merge attempts to read records from file $LOCAL.OLD. If file OLD does not exist, Sort/Merge performs a null sort or merge (no input records).

Remarks

- An input file name referenced without a file path is assumed to be in the working catalog unless the file name is for a standard system file. Standard system files, such as $INPUT or $OUTPUT, are assumed to be in the $LOCAL catalog.
- A single SORT or MERGE command can read as many as 100 input files. The files are read in the order specified on the command and on directives.
- More than one FROM parameter can be specified for a sort or merge: the command can specify one FROM parameter and each Sort/Merge directive can specify one FROM parameter.
- If a specified input file does not exist, Sort/Merge issues a warning error.
- Specifying FROM=$NULL indicates that no input files are specified. Assuming no owncode 1 procedure is specified, this results in a null sort or merge; a null sort or merge has no records sorted or merged.

- Sort/Merge does not read records from an input file past an embedded end-of-partition delimiter or the end-of-information.
- Sort/Merge input files can reside on mass storage or magnetic tape. For information on assigning file names to magnetic tape, see the NOS/VE System Usage manual.
- Sort/Merge input files can be have sequential, indexed-sequential, or direct-access file organization and variable-length (V), fixed-length (F), or trailing-character-delimited (T) record type.
- If an input file is an indexed-sequential file or direct-access file with a nonembedded key, the primary-key value is inserted at the beginning of the record when the record is read. Thus, position 1 is the first byte of the primary-key value and so key field and sum field definitions must be adjusted accordingly.
- Sort/Merge input files can be written using segment access as well as record access. However, Sort/Merge uses record access calls to open its input files; therefore, the records written using segment access must conform to the attributes of the file.
- For example, the record format must conform that that specified by the RECORD_TYPE attribute and the record length must not exceed the length specified by the MAX_RECORD_LENGTH attribute.

KEY (K)

Purpose Specifies one or more key field definitions.

Format *KEY=((key field definition* 1), ..(key_field_definition_2), ...)

Default:

A single key beginning at the first byte position of the record. Its length is the smallest minimum record length of the input files. Its type is ASCII and it is sorted in ascending order.

If the minimum MINIMUM_RECORD_LENGTH attribute for all input files is the default value zero, Sort/Merge uses 1 byte as the key length. If the minimum MINIMUM_RECORD_LENGTH attribute for all input files is greater than 1023 bytes, Sort/Merge uses 1023 bytes as the key length.

Remarks

- The KEY parameter can be specified once on the command and once on each directive.
- The key field definition order is the order that the defined keys are used. The first definition defines the major key; any subsequent definitions define minor keys. Sort key concepts are discussed in chapter 5.
- The total number of bytes in key fields cannot exceed 1023. The total number of key fields defined for a sort or merge cannot exceed 106. Sort/Merge issues a fatal error if either limit is exceeded.
- Sort/Merge allows key fields to overlap other key fields, except for key fields that are ordered by collating sequences defined with the alter option and key fields that overlap sum fields.
- If the output (TO) file is an indexed-sequential file, the major sort key must be the embedded primary key defined for the output file. For more information, see the TO parameter description.
- Key field definitions within the list are separated by a comma or a space. If more than one definition is specified, the list must be enclosed in parentheses.

NOTE

Be careful not to confuse the parentheses enclosing a key field definition with the parentheses enclosing the list of key field definitions. For example, the following parameters both define a single key that is 20 bytes long, starting in byte 6:

```
KEY=6..25      KEY=((6,20))
```

The following parameter does not define the same key; it defines a sort on two single-byte keys where byte 6 is the major key and byte 20 is the minor key:

```
KEY=(6,20)
```

LIST (L)

Purpose Specifies the file to which listing information is written.

Format *LIST=file*

Default:

If you omit the L parameter, listing information is written to file $LIST.

Remarks

- A listing file name referenced without a file path is assumed to be in the working catalog unless the file name is for a standard system file. Standard system files, such as $INPUT or $OUTPUT, are assumed to be in the $LOCAL catalog.
- If you specify the file $NULL with the L parameter, listing information is not written.
- Listing information includes the Sort/Merge version and level numbers, time and date, error messages, an exception file summary, and the number of records sorted or merged.
- The following are messages written to the listing file at the end of a sort or merge.

  If a catastrophic error occurred, the message is:

  ```
  CATASTROPHIC ERROR
  ```

  If one or more fatal errors occurred, the message is:

  ```
  FATAL ERROR(S)
  ```

  The message stating the number of records sorted or merged is:

  ```
  n records sorted/merged
  ```

  If records were written to the exception file, the following message summarizes the number (n) of records written to the exception records file:

  ```
  Exception file summary (n records written)
  ```

  At the end of a sort, this message is written on the job log:

```
MERGE COMPLETED
```

If Sort/Merge cannot complete the sort or merge request, the message written is one of the following:

```
SORT UNABLE TO COMPLETE
MERGE UNABLE TO COMPLETE
```

LIST_OPTIONS (LO)

Purpose Specifies the additional information written to the listing file.

Default:

LO=S (only the source and the minimum information is written to the listing file). The minimum information Sort/Merge writes to the file is the page heading, error messages, the exception records file summary, and the number of records sorted or merged.

Format *LIST_OPTIONS=(list of keyword value)*

The following keyword values request additional information. The LO list can specify more than one of these keywords in any order:

OFF — All listing information is suppressed.

NONE — Same as the OFF keyword.

S — Source (copies of all directives read by Sort/Merge).

DE — Detailed exception information. A message is written for each occurrence that causes a record to be written to the exception file.

The DE keyword value is valid only if you specify an exception records file; otherwise, an informational error is issued and messages are written only once per key, sum field, or file that causes records to be written to the exception records file.

RS — Record statistics for the records sorted or merged. The statistics are from the result array; a message is written for each element of the array except the first. The result array format is shown in table 2-2.

MS — Merge statistics for the records merged.

Remarks Specify either the keyword value OFF or NONE to indicate that no additional information is to be written to the listing file. If you specify a keyword value requesting no additional information, you cannot also specify a keyword value that requests additional information.

Table 2-2. Result Array Format

Array Element	Contents
1	Number of elements of results you want returned (0 through 17).
2	Number of records read from input files.
3	Number of records deleted by an owncode 1 procedure.
4	Number of records inserted by an owncode 1 procedure.
5	Number of records inserted by an owncode 2 procedure.
6	Number of records sorted or merged. (Does not include any records written to the exception records file or any zero-length records unless ZERO_LENGTH_RECORDS=PAD is selected.)
7	Number of records deleted by an owncode 3 procedure.
8	Number of records inserted by an owncode 3 procedure.
9	Number of records inserted by an owncode 4 procedure.
10	Number of records written to the exception records file.
11	Number of records deleted by an owncode 5 procedure.
12	Number of records combined by summing.
13	Number of records written to the output file.
14	Actual minimum record length of all input records.
15	Average record length. (Total record length divided by the total number of input records.)
16	Actual maximum record length of all input records.
17	Number of zero-length records removed from the sort or merge by the ZERO_LENGTH_RECORDS=DELETE option.
18	Number of duplicate records removed from the sort or merge by the OMIT_DUPLICATES=YES option.

LOAD_COLLATING_TABLE (LCT)

Purpose Loads a collation table, that is, a weight table that defines a collating sequence. The table may be a NOS/VE predefined collating table or a user-defined collation table in an object library.

Default:

Required to load a collation table; otherwise, the collating sequences available are the six Sort/Merge collating sequences and collating sequences defined by COLLATING_SEQUENCE_x parameters. For more information, see chapter 1.

Format *LOAD_COLLATING_TABLE=(key_type,table_name)*

key_type

Name to be used in a key field definition to specify the collating sequence produced by the collation table. The name cannot be the name of a predefined collating sequence or the name of a collating sequence you have already defined.

table_name

Name of a collation table (either a NOS/VE predefined collation table or a user-defined collation table in an object library).

The collation table must be loadable by PMP$LOAD and specify a value for each of the 256 ASCII character codes.

Remarks

- The LOAD_COLLATING_TABLE parameter can be specified once on the command and once on each directive.
- The total number of COLLATING_SEQUENCE_NAME and LOAD_COLLATING_TABLE parameters cannot exceed 100.
- LOAD_COLLATING TABLE is not used with the COLLATING_SEQUENCE_x parameters; it is an entirely separate means of specifying a collating sequence.

- NOS/VE supplies 11 predefined collation tables. To use one of the NOS/VE predefined collation tables, you specify the name of the predefined collation tables as the table_name.

- Unlike user-defined collation table modules, use of NOS/VE predefined collation tables does not require the addition of an object library to the program library list. For more information, see appendix E.

- After a LOAD_COLLATING_TABLE parameter associates a key type name with a collation table, the key type name can be used in a key field definition.

- For example, to use the predefined collation table OSV$EBCDIC to define the key type FULL_EBCDIC, you would specify this parameter:

```
LOAD_COLLATING_TABLE=(full_ebcdic, OSV$EBCDIC)
```

- Then to define the first 10 bytes of the record as a key field to be sorted in ascending order using the key type, you would specify this Sort/Merge parameter:

```
KEY=((1,10,full_ebcdic,a))
```

- You can use any collation table stored as a module in an object library file if you have permission to read the file. To use the module, you perform these steps:

  1. Add the object library to your program library list using a SET_PROGRAM_ATTRIBUTES command, such as:

```
set_program_attributes ..
  add_library=$user.object_library
```

  2. Specify the name of the module defining the collation table and the table_name. For example:

```
sort from=unsorted_file to=sorted_file ..
load_collating_table=(upper_lower, ..
case_insensitive), key=((1..24,upper_lower,d))
```

OMIT_DUPLICATES (OD)

Purpose Specifies whether Sort/Merge outputs only one record in each set of records with equivalent key values.

Default:

Duplicates are not omitted; equivalent key values are processed as specified by the OWNCODE_PROCEDURE_5, RETAIN_ORIGINAL_ORDER, or SUM parameter.

Format OMIT_DUPLICATES=boolean

TRUE, YES, or ON	Duplicates are omitted.
FALSE, NO, or OFF	Duplicates are not omitted.

Remarks

- Duplicate records are records that have equivalent key values.
- Each sort or merge can specify only one method of processing records with equivalent key values. Therefore, the OMIT_DUPLICATES, OWNCODE_PROCEDURE_5, RETAIN_ORIGINAL_ORDER, and SUM parameters are mutually exclusive.
- When duplicates are to be omitted, Sort/Merge removes the shorter duplicate records from the sort or merge. When the duplicates are the same length, any of the duplicates could be the one that is kept.
- A count is kept in word 18 of the result array of the number of duplicate records deleted from the sort or merge.
- Zero-length records are duplicates only if the ZERO_LENGTH_PARAMETER specifies the PAD option.

OWNCODE_FIXED_LENGTH (OWNFL or OFL)

Purpose Specifies the length of each fixed-length record entering a sort or merge from an owncode procedure.

Default:

The record length is specified by the OWNCODE_MAXIMUM_RECORD_LENGTH parameter or the largest MAXIMUM_RECORD_LENGTH attribute of the input or output files.

If you specify OWNCODE_PROCEDURE_1 and OWNCODE_PROCEDURE_3 parameters, but omit the FROM and TO parameters, you must specify either the OWNCODE_FIXED_LENGTH or OWNCODE_MAXIMUM_RECORD_LENGTH parameter.

Format *OWNCODE_FIXED_LENGTH = integer expression*

Remarks

- The record length can be from 1 through 65,535 bytes. Sort/Merge issues a fatal error for each record supplied whose length is not equal to the fixed record length.
- You cannot specify both the OWNCODE_FIXED_LENGTH and OWNCODE_MAXIMUM_RECORD_LENGTH parameters for the same sort.
- The OWNFL abbreviation is provided for CYBER 170 SORT5 compatibility; its continued use is not recommended.

OWNCODE_MAXIMUM_RECORD_LENGTH (OWNMRL or OMRL)

Purpose Specifies the maximum length of all variable-length records entering the sort or merge from an owncode procedure.

Default:

If you omit both the OWNCODE_MAXIMUM_RECORD_LENGTH and the OWNCODE_FIXED_LENGTH parameters, the input record length is the record length of the input and output files. If all input and output files have fixed-length records of the same length, this length is used. Otherwise, the largest maximum record length from any input or output file is used.

If you specify OWNCODE_PROCEDURE_1 and OWNCODE_PROCEDURE_3 parameters, but omit the FROM and TO parameters, you must specify either the OWNCODE_FIXED_LENGTH or OWNCODE_MAXIMUM_RECORD_LENGTH parameter.

Format *OWNCODE_MAXIMUM_RECORD_LENGTH = integer expression*

Remarks

- The maximum record length can be from 1 through 65,535 bytes.
- The integer must be large enough for all of the keys or else the sort order is undefined.
- Sort/Merge issues a fatal error if an owncode procedure supplies a record whose length is greater than the maximum record length.
- You do not need to specify a record length parameter if the sort has an input or output file with a maximum record length at least as long as the longest record supplied by an owncode procedure.
- You cannot specify both the OWNCODE_FIXED_LENGTH and OWNCODE_MAXIMUM_RECORD_LENGTH parameters for the same sort.

- The OWNMRL abbreviation is provided for CYBER 170 SORT5 compatibility; its continued use is not recommended.

OWNCODE_PROCEDURE_n (OPn or OWNn)

Purpose Specifies the name of an owncode procedure that is executed each time a certain event occurs during the sort or merge.

Default:

If you omit all OWNn parameters, no owncode procedures are executed.

Format *OWNCODE_PROCEDURE_n=name*

The suffix n is the digit 1, 2, 3, 4, or 5. The specified name is an entry point name in an object library.

NOTE

Owncode procedure names must be specified using uppercase letters only unless you specify C170_COMPATIBLE=YES.

Remarks

- Each sort or merge can specify only one method of processing records with equivalent key values. Therefore, the OWNCODE_PROCEDURE_5, OMIT_DUPLICATES, RETAIN_ORIGINAL_ORDER, and SUM parameters are mutually exclusive.
- You cannot specify an owncode 1 or an owncode 2 procedure for a merge.
- Owncode procedures are described in detail in chapter 3.
- The OWNn abbreviations are provided for CYBER 170 SORT5 compatibility; their continued use is not recommended.
- To make your owncode procedure available to Sort/Merge requires two steps: generation of an object library containing your owncode procedure and addition of the object library to the current object library list.

  To generate an object library, you use the CREATE_OBJECT_LIBRARY command utility. For more information, see the NOS/VE Object Code Management Usage manual.

To add the object library to the current object library list, enter a SET_PROGRAM_ATTRIBUTE command before the SORT or MERGE command. The SET_PROGRAM_ATTRIBUTE command is described in the NOS/VE Object Code Management Usage manual.

Chapter 4 contains an example showing the generation of an object library and its addition to the current object library list.

RESULT_ARRAY (RA or RESA)

Purpose Specifies an SCL array variable to be used as the result array.

Default:

If this parameter is omitted, the Sort/Merge statistics are not stored in an SCL variable. However, the statistics may be written to the listing file depending on the LIST_OPTIONS parameter value.

Format *RESULT_ARRAY=array name*

Remarks

- The result array is a single dimensional array of up to 18 integers. You set the first element of the result array to the number of elements in the result array to receive information (0 through 17).
- The SCL array variable must be defined before the SORT or MERGE command. For example, these commands create a variable and initialize its first element to 15:

```
create_variable, result_array, kind=integer ..
  dimension=1..16
result_array(1)=15
```

- The statistics returned in the array are listed in table 2-2.
- The RESA abbreviation is provided for CYBER 170 SORT5 compatibility; its continued use is not recommended.

RETAIN_ORIGINAL_ORDER (ROO or RETAIN or RET)

Purpose Specifies whether Sort/Merge is to output records with equivalent keys in the same order as the records are input.

Default:

The original order is not retained (records with equal sort key values are output in either order).

Format *RETAIN_ORIGINAL_ORDER = boolean*

An SCL boolean is a logical true or false value. A true specification (TRUE) indicates that the original order is to be retained; a false specification (FALSE) indicates that the original order need not be retained.

Remarks

- Each sort or merge can specify only one method of processing records with equivalent key values. Therefore, the RETAIN_ORIGINAL_ORDER, OMIT_DUPLICATES, OWNCODE_PROCEDURE_5, and SUM parameters are mutually exclusive.
- Maintaining the original order of records with equal key values increases the required processing time because Sort/Merge must keep track of the input order.
- If you specify more than one input file, the order you specify the files is the order the records with equal key values are output.
- The RETAIN and RET abbreviations are provided for CYBER 170 SORT5 compatibility; their continued use is not recommended.

STATUS

Purpose Specifies an SCL status variable in which the SORT or MERGE task returns its completion status.

Default:

None (the completion status is displayed at the terminal or written in the batch job log).

Format *STATUS=status variable*

Remarks

- The status variable is created before the sort or merge by the SCL command CREATE_VARIABLE. For example, the following command creates a status variable named SORT_STATUS:

```
create_variable, sort_status, kind=status
```

  For more information, see the NOS/VE Commands and Functions manual.

- A status variable is often used when the command is executed as part of a procedure. A fatal Sort/Merge error does not terminate the procedure. After command execution, the procedure should check the contents of the status variable to determine the next command executed.

- Sort/Merge treats errors of severity ERROR as fatal errors.

SUM (S)

Purpose Specifies that, when the sort or merge encounters two records having equal key values, the contents of the fields specified on the SUM parameter are to be summed and a single record written to the output file, replacing the two records with equal key values.

(Although the original records are removed from the sort or merge, they are not written to the exception records file.)

Default:

If you omit the SUM parameter, records are not summed.

Remarks The SUM parameter can be specified once on the command and once on each directive.

Sum Field Specification

The format of the SUM parameter is as follows:

```
SUM=((sum field definition 1), (sum_field_definition_2), ... )
```

Each sum field definition in the list is separated by a comma or a space. A sum field definition is a list of positional values that define a sum field using either of the following formats:

(**range**,*data_type*,*repeat_count*)

(**first**,*length*,*data_type*,*repeat_count*)

Only the first value is required; all additional values are optional.

The first value in the sum field definition is required. If the first value in the definition is a range, it defines the position and length of the sum field. The range specifies upper and lower integer bounds that are the first and last byte or bit positions in the sum field. For example, the range 10..15 defines a sum field from bit or byte position 10 through bit or byte position 15.

If the first value in the value set is not a range, the first value specifies the first position of the sum field and the optional second value specifies the field length. The default field length is 1.

Byte and bit positions in the record are numbered from the left beginning with 1. Sort/Merge interprets the key field location and length as bytes unless the data type is INTEGER_BITS or BINARY_BITS.

Table 2-3 lists the maximum sum field sizes.

The third value in the list specifies the data type. The default data type is INTEGER. All numeric formats, binary or character, are valid. See chapter 1 for the list of numeric data formats.

The fourth value in the sum field definition, the repeat count, is optional; the default value is 1. If specified, the SUM parameter defines contiguous sum fields within the record. Each sum field has the same length and contains the same type of numeric data.

For example, the following sum field definition defines three sum fields at byte positions 1 through 10, 11 through 20, and 21 through 30.

```
(1,10,BINARY,3)
```

Table 2-3. Maximum Sum Field Sizes

Numeric Format	Maximum Size (in bytes)	Numeric Format	Maximum Size (in bytes)
NUMERIC_FS	1023	BINARY	8
NUMERIC_LO	38	BINARY_BITS	8184 (bits)
NUMERIC_LS	38	INTEGER	8
NUMERIC_NS	38	INTEGER_BITS	8184 (bits)
NUMERIC_TO	38	PACKED	19
NUMERIC_TS	38	PACKED_NS	19
REAL	8 or 16		

Sum Field Rules

The new record contains the eqivalent key values in the key fields and the summed values in the sum fields. A data field that is not a key or sum field is written to the new record as the contents of the field in the longer of the two original records.

You can specify a maximum of 100 sum fields in a record. You can specify the SUM parameter more than once in a directive file.

Sum fields cannot overlap one another. Sum fields cannot overlap key fields.

Each sort or merge can specify only one method of processing records with equivalent key values. Therefore, the RETAIN_ORIGINAL_ORDER, OMIT_DUPLICATES, OWNCODE_PROCEDURE_5, and SUM parameters are mutually exclusive.

If a sum field contains no data because the input record is too short to include the field, a default value of zero in the appropriate format is used. The processing of partial sum fields and sum fields that contain invalid data is described later.

Sum Field Example

Suppose a university has two files of student information: a master file and an update file. The files contain identical information except that the master file contains information from past semesters while the update file contains only information for the current semester. Each record in both files contains these fields: student number, number of units registered, number of units completed, and grade points.

A new master file is generated by merging the two files. When the files are merged, the records having matching student numbers are summed and a new record written replacing the records from the master and update files. The new record in the output file contains the student number and the total number of units attempted, total number of units completed, and the total number of grade points.

Chapter 4 contains an example of SUM parameter use.

Exception Processing for Partial Sum Fields

Sort/Merge checks that each record is long enough to contain all defined sum fields when it reads the record. If an entire sum field is omitted, Sort/Merge sums the record as if it has a zero value in the field.

If the record contains a partial sum field, Sort/Merge processes it as an exception. The exception processing differs if an exception records file is specified:

If an exception records file is specified:

> Sort/Merge writes the record with the partial sum field to the exception records file. It writes the record with its original data as it was read from the input file. It then removes the record from the sort or merge.

If an exception records file is not specified:

> Sort/Merge keeps the record with the partial sum field in the sort or merge.

If Sort/Merge finds additional records whose key fields are equivalent to those of the record with the partial sum field, it sums the records as if the partial sum field contains a valid value; it does not process the partial sum field as invalid data. However, because the results of summing with a partial field are undefined, the resulting contents of the sum field are undefined.

If it reads any records with partial sum fields, Sort/Merge returns a summary diagnostic at the end of the sort or merge that gives the number of records with partial sum fields.

Exception Processing for Summing Errors

Sort/Merge detects summing errors when it attempts to sum fields. Only one error is detected per sum field. The summing error is processed as an exception.

If the LIST_OPTIONS parameter requests detailed error reporting (DE), Sort/Merge issues a diagnostic for each summing error.

The exception processing performed for summing errors depends on the error detected and whether an exception records file is specified for the sort or merge.

If an exception records file is specified:

1. Sort/Merge restores all sum fields of both records so the contents of each sum field are the same as before the summing of the records began.

2. If the error is due to invalid data or an indefinite real, Sort/Merge knows that at least one of the sum fields in the records is in error; it does not know whether the same sum field in the other record is also in error.

3. Therefore, Sort/Merge writes the record it knows to be in error to the exception records file and removes it from the sort or merge, but leaves the other record in the sort or merge.

If Sort/Merge detects an arithmetic overflow or underflow error or finds that each record has invalid data in different sum fields, it knows that both rcords are in error. Therefore, it writes both records to the exception records file and removes both from the sort or merge.

If an exception records file is not specified:

1. Sort/Merge deletes one of the records. If one record is longer than the other, the shorter record is deleted. Otherwise, either record could be deleted.

2. The other record remains in the sort or merge with undefined data in the sum field for which the error was detected. Summing is completed for the other sum fields.

TO (T)

Purpose Specifies the output file to which sorted or merged records are written (if records are left after owncode procedure processing).

Format *TO=file*

Default:

If you omit the TO parameter but specify the OWNCODE_PROCEDURE_3 parameter, Sort/Merge assumes the owncode 3 procedure performs output processing. If the owncode 3 procedure passes records back to Sort/Merge, Sort/Merge writes the records on file NEW. If file NEW does not exist, Sort/Merge creates file NEW in the $LOCAL catalog.

If you omit the TO parameter and the OWNCODE_PROCEDURE_3 parameter, Sort/Merge writes all output records to file NEW. If file NEW does not exist, Sort/Merge creates file NEW in the $LOCAL catalog.

Remarks

- An output file name referenced without a file path is assumed to be in the working catalog unless the file name is for a standard system file. Standard system files, such as $INPUT or $OUTPUT, are assumed to be in the $LOCAL catalog.
- The output file cannot also be an input file.
- If you specify the file $NULL with the TO parameter, sorted or merged records are not written to a file.
- Sort/Merge writes records to the output file using the system procedure AMP$PUT_NEXT. If AMP$PUT_NEXT returns an error for a record, Sort/Merge writes the record to the exception records file instead (if one is specified).
- The Sort/Merge output file can reside on either mass storage or magnetic tape. For more information on assigning file names to magnetic tape, see the NOS/VE System Usage manual.

- The output (TO) file cannot be a direct-access file.

  If the output (TO) file is a direct-access file, Sort/Merge issues a fatal error. If appropriate, use the COPY_KEYED_FILE command to convert the TO file to a direct-access file.

- If the outut file is an indexed-sequential file with a nonembedded primary key, the primary-key value is removed from the beginning of each output record before the record is written. The primary-key value is stored in the primary index. The record data is shortened by key-length bytes.

- If the output file is an indexed-sequential file, the major sort key must be the primary key for the file. Thus, the major sort key value for each input record must be unique because the indexed-sequential file origanization requires unique primary-key values. This can be ensured by specifying the OMIT_DUPLICATES=YES parameter or using an owncode 5 procedure.

- If the output (TO) file is an indexed-sequential file, Sort/Merge checks the KEY_POSITION, KEY_LENGTH, and KEY_TYPE attributes:

  - If the major sort key position does not match the KEY_POSITION attribute value, Sort/Merge issues a fatal error and terminates.

  - If the major sort key length does not match the KEY_LENGTH attribute value, Sort/Merge issues a warning error and changes the major sort key length to match the primary key length.

  - If the major sort key type does not match the KEY_TYPE attribute value, Sort/Merge issues a warning error and changes the major sort key type if the KEY_TYPE value is UNCOLLATED or INTEGER. (It does not issue a warning or change the key type if the KEY_TYPE value is COLLATED.)

    - If the KEY_TYPE is UNCOLLATED, the major sort key type is changed to ASCII.

- If the KEY_TYPE is INTEGER, the major sort key type is changed to INTEGER.

– You can define the output file attributes using a SET_FILE_ATTRIBUTES command. To read about indexed-sequential file attributes, see the discussion of keyed file creation in chapter 5.

VERIFY_MERGE_INPUT_ORDER (VMIO or VERIFY or VER)

Purpose Specifies whether Sort/Merge checks the order of the merge input records. The records must be in sorted order.

Default:

If the VERIFY_MERGE_INPUT_ORDER parameter is omitted, the record order is not checked.

Format *VERIFY_MERGE_INPUT_ORDER = boolean*

An SCL boolean is a logical true or false value. A true specification (TRUE) indicates that the order of the merge input records is to be checked; a false specification (FALSE) indicates the order of the merge input is not to be checked.

Remarks

- If the records in the merge input files are not pre-sorted on the sort keys for the merge, the out-of-order records remain out-of-order in the merge output file. The order of the out-of-order records is undefined.
- If, while verifying the record order, Sort/Merge encounters a record out of order, it issues a warning message and continues merging.
- If an exception records file was specified for the merge, the out-of-order record is written to the exception records file and deleted from the merge operation. If an exception records file was not specified, the out-of-order record is kept in the merge; its position within the output file is undefined.
- If you specify the VERIFY_MERGE_INPUT_ORDER parameter on a sort, Sort/Merge issues a warning message but otherwise ignores the parameter.
- The VERIFY and VER abbreviations are provided for CYBER 170 SORT5 compatibility; their continued use is not recommended.

ZERO_LENGTH_RECORDS (ZLR)

Purpose Specifies the disposition of zero-length input records.

NOTE

This parameter applies only to records read from input files; it does not apply to records supplied by owncode procedures.

Default:

DELETE.

Format *ZERO_LENGTH_RECORDS=keyword*

One of the following keywords specifying the disposition of all zero-length input records read for the sort or merge:

DELETE	Each zero-length record is deleted from the sort or merge. It is not written to the exception records file.
PAD	Each zero-length record is processed as a short record. For more information, see Short Records in chapter 1.
LAST	Each zero-length record is written at the end of the output.

Remarks For more information, see Zero-Length Records in chapter 1.

Owncode Procedures 3

Owncode Procedures 3

You can write subprograms to insert, substitute, modify, or delete input and output records during Sort/Merge processing. Such a subprogram, called an owncode procedure, is executed each time the sort or merge reaches a certain point in Sort/Merge processing. The points at which owncode procedures are called are listed below:

Sorts Only:

Owncode 1 After an input record is read.

Owncode 2 After the end of an input file is read.

Sorts or Merges:

Owncode 3 Before an output record is written.

Owncode 4 After the output file is written.

Owncode 5 When two records are compared and found to have equivalent key values.

Sort/Merge passes a record to the owncode procedure, which processes the record. When the record is returned to Sort/Merge from the owncode procedure, Sort/Merge processes the record according to a code specified by the owncode procedure.

Owncode procedures can also supply the records to be sorted. When Sort/Merge is ready for a record, it calls the owncode 1 procedure which then passes a record to Sort/Merge.

Owncode procedures are written in a programming language such as FORTRAN (subroutine subprograms), COBOL (subprograms compiled with COBOL SP=TRUE option), CYBIL, or any other language that uses the standard calling sequence. CYBIL owncode procedures must be declared as XDCL procedures.

Owncode procedure must be compiled and saved as load modules on object libraries. Object libraries are created using the CREATE_OBJECT_LIBRARY utility as described in the NOS/VE Object Code Management Usage manual.

To use an owncode procedure in a sort or merge, the procedure must be loadable from the program library list. You can add object libraries to the program library list using the SET_PROGRAM_ATTRIBUTES command.

Chapter 4 contains an example of storing an owncode procedure in an object library.

Owncode Procedure Parameters

Sort/Merge communicates with an owncode procedure via the procedure parameter list. Sort/Merge passes record data to the procedure and the procedure returns record data and a code indicating how Sort/Merge is to process the record data.

Table 3-1 summarizes the parameters passed between Sort/Merge and owncode procedures.

Table 3-1. Owncode Procedure Parameters

Parameter	Description
return_code	Integer code set by the owncode procedure and returned to Sort/Merge
reca	Array containing record data
rla	Integer length of the reca record
recb	Array containing record data for second record (used only by an owncode 5 procedure)
rlb	Integer length of the recb record (used only by an owncode 5 procedure)

The return_code parameter passes an integer code back to Sort/Merge specifying how Sort/Merge is to process the returned record. Sort/Merge always initializes the return_code value to 0 when it calls an owncode procedure.

The owncode procedure can leave the return_code value unchanged or change it to one of the valid values for the owncode procedure. (The valid values are listed in the individual owncode procedure descriptions.) If an invalid return_code value is returned, Sort/Merge returns a fatal error.

The subsequent parameters are used to pass one or two records to the owncode procedure. For an owncode 1 through owncode 4 procedure, Sort/Merge passes only one record, the curent record being input or output. The record data is passed in the reca variable and the record length in bytes in the rla variable.

When calling an owncode 5 procedure, Sort/Merge passes two records having equivalent key values. The record data is passed in the reca and recb variables and the corresponding record lengths in the rla and rlb variables, respectively.

An owncode procedure can change the record data and record length values passed to it. However, the procedure must ensure that the correct record length is returned for the record data.

Owncode Record Length

Sort/Merge checks the length of each record returned to it by an owncode procedure. If a record is too long, Sort/Merge issues an error.

The Sort/Merge specification can explicitly specify the owncode record length. Otherwise, by default, the maximum record length is the largest MAXIMUM_RECORD_LENGTH attribute value of the input and output files specified for the sort or merge.

To explicitly specify the owncode record length, use the OWNCODE_FIXED_LENGTH or OWNCODE_MAXIMUM_RECORD_LENGTH parameter. If the sort specifies owncode 1 and owncode 3 procedures but no input or output files, a parameter to specify the owncode record length is required.

If you specify OWNCODE_FIXED_LENGTH, the record length returned by an owncode procedure must exactly match the specified record length value. If you specify OWNCODE_MAXIMUM_RECORD_LENGTH, each record length returned cannot exceed the specified record length value.

Owncode 1: Processing Input Records

You specify an owncode 1 procedure to process or supply the input records for a sort. An owncode 1 procedure is used only with a sort; specifying an owncode 1 procedure for a merge returns a fatal error.

Owncode 1 procedure processing varies depending on whether input files are specified for the sort.

One or More Input Files Specified

If you specify one or more input files for a sort (even if the input file is $NULL), Sort/Merge calls the owncode 1 procedure each time it rads an input record. Sort/Merge passes the input record in the reca variable, the record length in the rla variable, and the return_code variable initialized to 0.

After owncode processing of the record, control returns to Sort/Merge, which processes the record passed back in reca according to the return_code value set by the owncode 1 procedure. The contents of the reca and rla variables can differ from those originally passed to the procedure.

The following are the valid return_code values and their meanings:

0 Sort/Merge sorts the record passed back in reca and reads the next input record.

1 Sort/Merge does not sort the record in reca and reads the next input record.

2 Sort/Merge sorts the record passed back in reca, but does not read the next input record. Instead, Sort/Merge calls the owncode 1 procedure again so additional records can be added to the sort. The owncode 1 procedure should continue to specify return_code 2 until all records to be inserted at this point have been passed; it should then set the return_code to 0.

3 Sort/Merge does not sort the record passed back in reca. It closes the current input file and calls the owncode 2 procedure (if any). After owncode 2 procedure processing, it opens the next input file (if any) and reads the next input record.

For example, to insert one record after the current input record, the owncode 1 procedure performs the following steps:

1. Checks that the record passed in reca is the record after which the new record is to be inserted.

2. Sets the return_code value to 2 and returns control to Sort/Merge.

3. When called again, it stores the new record in reca, stores the length of the new record in rla, sets the return_code value to 0, and returns control to Sort/Merge.

Input Files Not Specified

If you do not specify any input files for the sort (the FROM parameter is omitted), Sort/Merge calls the owncode 1 procedure as the source of input records. Sort/Merge passes reca as an empty array of the maximum record length, rla set to 0, and the return_code value initialized to 0.

The following are the valid return_code values and their meanings:

0 Sort/Merge sorts the record passed back in reca, clears the reca array, sets the rla and return_code variables to 0, and calls the owncode 1 procedure again.

2 Sort/Merge sorts the record passed back in reca, leaves the data in reca and the record length in rla, initializes the return_code to 0, and calls the owncode 1 procedure again.

3 Sort/Merge does not sort the record passed back in reca and calls the owncode 2 procedure if one has been specified; otherwise, it terminates the input process.

Owncode 2: Processing Input Files

You specify an owncode 2 procedure to supply input at the end of each input file. An owncode 2 procedure is used only with a sort; specifying an owncode 2 procedure for a merge returns a fatal error.

Owncode 2 procedure processing varies depending on whether input files are specified for the sort.

One or More Input Files Specified

If you specify one or more input files for the sort (even if the input file is $NULL), Sort/Merge calls the owncode 2 procedure when it terminates input. It terminates input when it reads an end-of-partition delimiter, or the end-of-information, or receives a return_code value of 3 from an owncode 1 procedure.

Sort/Merge passes reca as an empty array of the maximum record length, rla set to 0, and the return_code variable initialized to 0.

The following are the valid return_code values and their meanings:

0 Owncode 2 processing ends; Sort/Merge opens the next input file, if any, and reads the next input record.

1 Sort/Merge sorts the record passed back in reca and calls the owncode 2 procedure again.

For example, to insert one record at the end of an input file, the owncode 2 procedure performs the following steps:

1. Stores the record in reca, stores the record length in rla, sets the return_code value to 1 and returns control to Sort/Merge.
2. When called again, it leaves the return_code value set to 0 and returns control to Sort/Merge.

Input Files Not Specified

If you do not specify any input files for the sort (the FROM parameter is omitted), Sort/Merge calls the owncode 2 procedure after the owncode 1 procedure returns a return_code value of 3.

Sort/Merge passes reca as an empty array of the maximum record length, rla set to 0, and the return_code value initialized to 0.

The following are the valid return_code values and their meanings:

0 Owncode 2 processing ends, signaling the end of the input records for the sort.

1 Sort/Merge sorts the record passed back in reca and calls the owncode 2 procedure again.

Owncode 3: Processing Output Records

You specify an owncode 3 procedure to process output records from a sort or merge.

Owncode 3 procedure processing varies depending on whether an output file is specified for the sort or merge.

Output File Specified

If you specify an output file for the sort or merge (even if it is $NULL), Sort/Merge calls the owncode 3 procedure each time an output record is ready to be written. Sort/Merge passes the output record to the procedure in the reca variable, the record length in bytes in the rla variable, and the return_code variable initialized to 0.

After owncode processing of the record, control returns to Sort/Merge, which processes the record passed back in reca according to the return_code value set by the owncode 3 procedure. The contents of the reca and rla variables can differ from those originally passed to the procedure.

The following are the valid return_code values and their meanings:

0 Sort/Merge writes the record passed back in reca to the output file. It then passes the next output record, if any, to the owncode 3 procedure.

1 Sort/Merge does not write the record passed back in reca to the output file. It passes the next output record, if any, to the owncode 3 procedure.

2 Sort/Merge writes the record passed back in reca to the output file, leaves the data in reca and the record length in rla, initializes the return_code to 0, and calls the owncode 3 procedure again.

3 Sort/Merge does not write the record passed back in reca. It calls the owncode 4 procedure if one is specified; otherwise, it terminates the sort or merge.

For example, to insert one record after the current output record, the owncode 3 procedure performs the following steps:

1. Checks that the record passed in reca is the record after which the new record is to be inserted.
2. Sets the return_code value to 2 and returns control to Sort/Merge.
3. When called again, it stores the new record in reca, stores the length of the new record in rla, sets the return_code value to 0, and returns control to Sort/Merge.

Output File Not Specified

If you do not specify an output file (you omit the TO parameter), the owncode 3 procedure performs all processing of output records. Sort/Merge passes each output record to the owncode 3 procedure, but it does not process any record returned by the procedure. Sort/Merge does not write any output records.

Sort/Merge passes the output record to the procedure in the reca variable, the record length in bytes in the rla variable, and the return_code variable initialized to 0.

The following are the valid return_code values and their meanings:

0 Sort/Merge calls the procedure again, passing the next output record.

1 Sort/Merge calls the procedure again, passing the next output record.

2 Sort/Merge calls the procedure again, passing the same output record.

3 Sort/Merge terminates the output process, even if it has additional output records. It then calls the owncode 4 procedure if one has been specified; otherwise, it terminates the sort or merge.

Owncode 4: Processing the Output File

You specify an owncode 4 procedure to write additional output records to the end of the output file. An owncode 4 procedure can be used with a sort or a merge.

Owncode 4 procedure processing varies depending on whether an output file is specified for the sort or merge.

Output File Specified

If you specify an output file for the sort or merge (even if it is $NULL), Sort/Merge calls the owncode 4 procedure after it has written its last output record to the output file.

Sort/Merge passes reca as an empty array of the maximum record length, rla set to 0, and the return_code initialized to 0.

The following are the valid return_code values and their meanings:

0 Sort/Merge terminates the sort or merge without writing the record passed back in reca.

1 Sort/Merge writes the record passed back in reca and calls the owncode 4 procedure again.

Output File Not Specified

An owncode 4 procedure cannot supply additional output records when no output file has been specified. Still, if you specify an owncode 4 procedure for a sort or merge without an output file, Sort/Merge calls the owncode 4 procedure after the owncode 3 procedure (if any) has terminated output.

The following are the valid return_code values and their meanings:

0 Sort/Merge terminates the sort or merge.

1 Sort/Merge terminates the sort or merge.

Owncode 5: Processing Records With Equal Keys

When an owncode 5 procedure is specified, Sort/Merge calls the owncode 5 procedure each time it compares the key values of two records and finds that the values are equivalent. It passes both records to the owncode 5 procedure for processing.

NOTE

Sort/Merge can interpret character key values that are not identical as equivalent . When the collating sequence used for the key assigns the same collating weight to more than one character, those characters are equivalent key values.

An owncode 5 procedure cannot be used when the OMIT_DUPLICATES, RETAIN_ORIGINAL_ORDER, or SUM parameter is specified for the sort or merge. A sort or merge can use only one method of processing records with equivalent key values.

For a given number (n) of records with equivalent key values, each record is passed to the owncode 5 procedure log n times. The order in which the records are passed is not defined.

NOTE

An owncode 5 procedure can change the record data passed to it, but it must not change the data in the key fields of the record. If it does, the sort order of the modified key fields is undefined.

The following are the valid return_code values and their meanings:

0 Sort/Merge accepts the first rla bytes of reca as the first record and the first rlb bytes of recb as the second record.

1 Sort/Merge accepts the first rla bytes of reca as the first record and deletes recb from the sort or merge.

2 Sort/Merge accepts the first rlb bytes of recb as the first record and the first rla bytes of reca as the second record.

3 Sort/Merge accepts the first rlb bytes of recb as the first record and deletes reca from the sort or merge.

4 Sort/Merge deletes both records from the sort or merge.

5 Sort/Merge does not read the record data returned by the procedure; it processes the two records in their original order (reca before recb).

6 Sort/Merge does not read the record data returned by the procedure, but it deletes the second record (recb) from the sort or merge.

7 Sort/Merge does not read the record data returned by the procedure, but it reverses the order of the two records (recb before reca).

8 Sort/Merge does not read the record data returned by the procedure, but it deletes the first record (reca) from the sort or merge.

For Better Performance

When the owncode 5 procedure does not change the record data, it should use return_code values 5, 6, 7, or 8 instead of return_code values 0, 1, 2, or 3. Performance is improved because Sort/Merge does not read the returned record data.

Do not use return_code 0 to reverse the order of the two records by exchanging the contents of reca and recb. Performing an exchange sort is both incompatible with and much slower than the Sort/Merge sorting algorithm.

If the owncode 5 procedure sorts the two records using one or more keys in addition to those specifed for the sort or merge, the procedure should use return_code values 5 and 7 only. (Return_code values 0 and 2 could also be used, but performance would be slower.)

Examples 4

Examples 4

This chapter contains examples of sorts and merges. The examples are as follows:

Command sort on one key

Command sort on multiple keys

Command merge

Using a directive file

Creating an object library

Summing a file

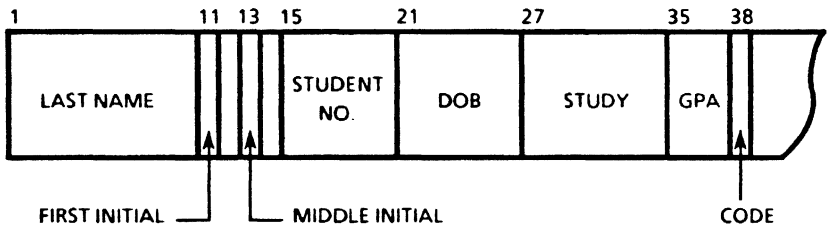
Defining your own collating sequence

NOTE

File names referenced without a file path are assumed to be in the working catalog unless the file name is for a standard system file. Standard system files, such as $INPUT or $OUTPUT, are assumed to be in the $LOCAL catalog.

Command Sort on One Key

The record layout of a university student file named UNIVERSITY_STUDENTS is shown below.



Each record includes the last name and first and middle initials, the student number, the date of birth, the field of study, the grade point average, and a code representing class (4=freshman, 3=sophomore, 2=junior, 1=senior); all fields contain character data. The file is maintained with the student number as the major key. Records are ordered in ascending order according to the student number as follows in file UNIVERSITY_STUDENTS.

```
WALLACE     S T  873661102 55ENGIR    2861
JOHNSON     M J  902480630 51MATH     2253
SANDERS     G R  998550228 58BUS      3011
NEECE       M L  999111213 58ART      2291
TERRELL     T H  999980403 56ENG      3861
OKADA       N A  100103111 750UNDEC   2225
REYES       S L  100246031 558ANTHRO  3341
SUGARMAN    B T  100528070 457SOC     3501
PHILLIPS    A D  100531121 158EDU     2112
KRUTZ       S T  100532010 353POLISCI 1981
SMITH       C R  100610103 058MATH    3791
  :
YEH         F L  102005120 645ART     2764
WARNES      D V  102116060 861POLISCI 2814
CARLSON     M K  102126022 355ENGIR   3454
FUHRMAN     L W  102212111 859CHEM    3204
MCMAHON     M C  102223061 260ENG     2784
JUNG        G D  102301052 561PHYSED  2214
POPOVICH    H W  102311100 961BUS     2434
JONES       J A  102318081 555EDU     2844
```

The command for sorting file UNIVERSITY_STUDENTS to generate an alphabetic list of students is:

```
SORT,FROM=UNIVERSITY_STUDENTS,TO=SORTED_FILE,KEY=1..10
```

The SORT command calls for records from UNIVERSITY_STUDENTS to be sorted in ascending order on a key that occupies character positions 1 through 10 in each record according to the default ASCII collating sequence. Sorted records are written to file SORTED_FILE, which is created as a local file during the sort. The contents of SORTED_FILE output from the sort is shown below.

```
BARTLETT  S S 100800100957ART      2735
BILLINGS  C Y 101579111855MUS      2965
CARLSON   M K 102126022355ENGIR    3454
CHARLES   S H 101418032459ANTHRO   2453
CLARK     D V 101023101956ENG      2083
CLARK     D N 101400102954ECON     3782
COCHRAN   G L 100725111857BIO      3011
DAVIES    E D 100812080656JOURN    2031
DAVIS     D A 100972071650ENR      3541
  :
WALLIN    G E 101056041659POLISCI  3151
WARNES    D V 102116060861POLISCI  2814
WILSON    W L 101967010261MATH     3454
WONG      S T 101001012755PSYCH    2152
WOO       R M 101315100159BUS      3223
WOODSTOCK C T 101497030160CHEM     3483
YEH       F L 102005120645ART      2764
YOST      D L 100880111158ENG      2582
ZEITZ     F K 100963111858MATH     2612
ZIMMERS   C A 101075063059MATH     2992
```

Command Sort on Multiple Keys

The command for sorting file UNIVERSITY_STUDENTS on three keys is:

```
SORT,FROM=university_students,TO=field_of_study,...
  KEY=((27..34),(38,1),(1..10))
```

Another way of specifying the sort is as follows:

```
SORT,FROM=university_students,TO=field_of_study,...
  KEY=(27..34,38,1..10)
```

The SORT command calls for records to be first sorted on the field of study (key number 1), which occupies character positions 27 through 34 in each record. Records with equal keys for the major key are then sorted on the class code (key number 2), which is a 1-character field in position 38. The third key sorts students with the same field of study and class by their last name (key number 3).

The commands also illustrate continuing the SORT command beyond one line. The first line of the command ends with three periods, indicating continuation. The second line contains the KEY parameter. Sorted records are written to the file FIELD_OF_STUDY shown below.

```
REYES      S L 100246031558ANTHRO  3341
MAYER      M I 100991122359ANTHRO  2882
CHARLES    S H 101418032459ANTHRO  2453
MARTIN     R C 100955082157ART     2891
NEECE      M L  99911121358ART     2291
NAKAMURA   S L 101529051260ART     2594
YEH        F L 102005120645ART     2764
BARTLETT   S S 100800100957ART     2735
COCHRAN    G L 100725111857BIO     3011
  :
KRUTZ      S T 100532010353POLISCI 1981
WALLIN     G E 101056041659POLISCI 3151
WARNES     D V 102116060861POLISCI 2814
WONG       S T 101001012755PSYCH   2152
LANGDON    M A 101754080549PSYCH   2013
LASEUR     P T 100678042256PSYCH   2233
SUGARMAN   B T 100528070457SOC     3501
SMITH      F R 101062120758SOC     2913
DOUGLAS    M L 101325071558UNDEC   2585
OKADA      N A 100103111750UNDEC   2225
```

Command Merge

The file ADD_STUDENTS is ordered according to the student number as shown below.

File ADD_STUDENTS:

```
QUINTERA   L S  90154101253BIO      3451
KING       M L 100012090848BUS      2431
ANDRUS     J R 100478042855JOURN    2121
UNGERMAN   J M 100933120356PHYSED   3012
KLEIN      S A 100987051260ENGIR    2762
IRVING     W R 101750111855ENG      3943
ALLEN      M G 102056012561LNGUIS   3854
GREENWOOD  M R 102168101961EDU      2264
ANDERSEN   C R 102308032160POLISCI  2544
EBERHARD   N I 102320061158BUS      3014
GOMEZ      J R 102379022260COMPSCI  2984
```

The files UNIVERSITY_STUDENTS and ADD_STUDENTS can be merged because they are sorted on the same key. The command to merge the two files is:

```
MERGE,FROM=(university_students,add_students),...
  TO=new_students_file,KEY=((15..20,NUMERIC_NS))
```

A new file (NEW_STUDENTS_FILE) is created as a permanent file to which merged records are written.

The MERGE command names the two input files and the new output file. The student number is the key on which the files are merged (the field on which the files are presorted). This field is numeric character data with leading blanks. The file NEW_STUDENTS_FILE output from the merge is shown below.

```
WALLACE    S T  87366110255ENGIR   2861
QUINTERA   L S  90154101253BIO     3451
JOHNSON    M J  90248063051MATH    2253
SANDERS    G R  99855022858BUS     3011
NEECE      M L  99911121358ART     2291
  :
TERRELL    T H  99998040356ENG     3861
KING       M L 100012090848BUS     2431
OKADA      N A 100103111750UNDEC   2225
REYES      S L 100246031558ANTHRO  3341
ANDRUS     J R 100478042855JOURN   2121
```

Using a Directive File

The contents of three directive files are shown below.

Directive File DIR_FILE_1:

```
SORT,FROM=UNIVERSITY_STUDENTS
SORT,TO=SORTED_STUDENT_NAMES
SORT,KEY=1..10
```

Directive File DIR_FILE_2:

```
SORT,FROM=ADD_STUDENTS
SORT,TO=SORTED_ADDED_STUDENTS
SORT,KEY=1..10
```

Directive File DIR_FILE_3:

```
MERGE,FROM=(SORTED_STUDENT_NAMES,SORTED_ADDED_STUDENTS)
MERGE,TO=MERGED_STUDENT_NAMES
MERGE,KEY=1..10
```

File DIR_FILE_1 sorts the student's names from the file UNIVERSITY_STUDENTS in ascending order, creating a new file called SORTED_STUDENT_NAMES. File DIR_FILE_2 sorts the student's names from the file ADD_STUDENTS in ascending order, creating a new file SORTED_ADDED_STUDENTS. Finally, file DIR_FILE_3 merges the files SORTED_STUDENT_NAMES and SORTED_ADDED_STUDENTS according to the student's name in ascending order, creating a new file called MERGED_STUDENT_NAMES.

The commands to call the three directive files are:

```
SORT,DIR=dir_file_1
SORT,DIR=dir_file_2
MERGE,DIR=dir_file_3
```

The contents of the file MERGED_STUDENT_NAMES output from the directive file sorts and merge is shown below. The records are in alphabetic order.

```
BARTLETT   S S 100800100957ART      2735
BILLINGS   C Y 101579111855MUS      2965
BRISCOE    J H 102343121157ENVIRO   2544
CARLSON    M K 102126022355ENGIR    3454
CHARLES    S H 101418032459ANTHRO   2453
CLARK      D V 101023101956ENG      2083
CLARK      D N 101400102954ECON     3782
COCHRAN    G L 100725111857BIO      3011
DAVIES     E D 100812080656JOURN    2031
DAVIS      D A 100972071650ENR      3541
  :
WALLIN     G E 101056041659POLISCI  3151
WARNES     D V 102116060861POLISCI  2814
WILSON     W L 101967010261MATH     3454
WONG       S T 101001012755PSYCH    2152
WOO        R M 101315100159BUS      3223
WOODSTOCK  C T 101497030160CHEM     3483
YEH        F L 102005120645ART      2764
YOST       D L 100880111158ENG      2582
ZEITZ      F K 100963111858MATH     2612
ZIMMERS    C A 101075063059MATH     2992
```

Creating an Object Library

You must place an owncode procedure into an object library before using it in a sort or merge, as detailed in chapter 3. A FORTRAN subroutine named OWNCODE that can be used as an owncode 3 procedure is shown below. The suboutine deletes the first record in a file. The variable COUNT keeps track of the number of times the owncode procedure is entered.

```
SUBROUTINE OWNCODE (retcode,reca,rla)
INTEGER retcode, rla, count
CHARACTER reca*38
DATA count /0/
count = count +1
IF (count.eq.1) THEN
   retcode = 1
ELSE
   retcode = 0
ENDIF
RETURN
END
```

For detailed information on placing a compiled subroutine into a library, see the NOS/VE Object Code Management Usage manual. Assuming the source text for the OWNCODE subroutine is on file $USER.OWNCODE, the commands to place OWNCODE into an object library on file $USER.OWN_LIBRARY are shown below.

```
/fortran input=$user.owncode
/create_object_library
COL/add_module library=$local.lgo
COL/generate_library library=$user.own_library
COL/quit
/display_object_library library=$user.own_library ..
../display_option=entry_point

OWNCODE                          - load module

entry points
~~~~~~~~~~~~

  OWNCODE

/set_program_attribute add_library=$user.own_library
```

After executing these commands, a SORT command such as the following can use the OWNCODE subroutine:

```
sort from=university_students to=results key=1..10 ..
 owncode_procedure_3=OWNCODE
```

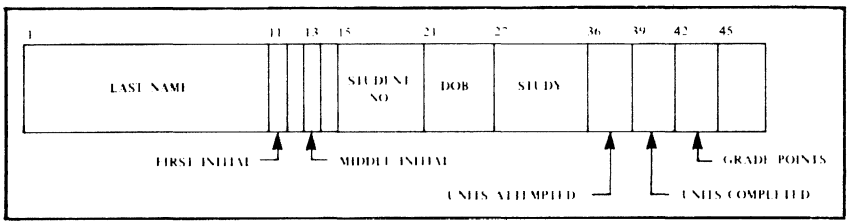
After the SORT command is executed, the file UNIVERSITY_STUDENTS is sorted, with the first record deleted. The sorted records are written to the file RESULTS as shown below.

```
BILLINGS   C Y 101579111855MUS      2965
BRISCOE    J H 102343121157ENVIRO   2544
CARLSON    M K 102126022355ENGIR    3454
CHARLES    S H 101418032459ANTHRO   2453
CLARK      D N 101400102954ECON     3782
CLARK      D V 101023101956ENG      2083
COCHRAN    G L 100725111857BIO      3011
DAVIES     E D 100812080656JOURN    2031
DAVIS      D A 100972071650ENR      3541
  :
WALLIN     G E 101056041659POLISCI  3151
WARNES     D V 102116060861POLISCI  2814
WILSON     W L 101967010261MATH     3454
WONG       S T 101001012755PSYCH    2152
WOO        R M 101315100159BUS      3223
WOODSTOCK  C T 101497030160CHEM     3483
YEH        F L 102005120645ART      2764
YOST       D L 100880111158ENG      2582
ZEITZ      F K 100963111858MATH     2612
ZIMMERS    C A 101075063059MATH     2992
```

Note that the owncode procedure has deleted the first record in the file.

Summing Records

The record layout of a university student file named STUDENTS is shown below.



Each record contains three numeric fields. They are: number of units attempted, number of units completed, and grade points. The file STUDENTS is shown below with multiple records for each student.

```
GREENWOOD M R 102168101961EDU      002002000
IRVING    W R 101750111855ENG      004004016
GREENWOOD M R 102168101961EDU      003003009
IRVING    W R 101750111855ENG      098095375
QUINTERA  L S  90154101253BIO      003000000
ALLEN     M G 102056012561LNGUIS   005000000
ALLEN     M G 102056012561LNGUIS   025020077
ALLEN     M G 102056012561LNGUIS   004004012
```

Records are to be sorted according to the student number. Using the SUM parameter, records with the same student number are combined into one record by adding the numeric fields together. The new record will give the total number of units attempted, total number of units completed, and the total number of grade points.

The SORT command to sort and sum the file STUDENTS is as follows:

```
SORT, FROM=students, TO=summed_file, KEY=(15..20),...
  SUM=((36,3,numeric_ns,3))
```

The input file STUDENTS is named, and the output file SUMMED_FILE will contain the results of the summing. The student number (positions 15 through 20) is specified as the sort key. The SUM parameter specifies that a three-position numeric field of type NUMERIC_NS begins in position 36 in each record. The repetition indicator specifies that three contiguous fields are to be summed. The

output from the sort is shown below. Each record ends with nine digits: the first three digits are the total units attempted, the next three are the total units completed, and the final three are the total grade points.

```
QUINTERA  L S  90154101253BIO      003000000
IRVING    W R  101750111855ENG      102099391
ALLEN     M G  102056012561LNGUIS   034024089
GREENWOOD M R  102168101961EDU      005005009
```

The output file contains one record for each student. The numeric fields are the totals of the units attempted, units completed, and grade points.

Defining Your Own Collating Sequence

The file BIRTHDATES, ordered according to the student name, is shown below. The file contains the students' last names, students' first and middle initials, and the students' dates of birth.

```
ALLEN       M G 10-09-61
ANDERSEN    C R 05-01-60
EBERHARD    N I 06 05 58
GREENWOOD   M R 09-12-61
IRVING      W R 01/07/55
KING        M L 11 11 48
QUINTERA    L S 08/12/53
WALLACE     S T 12/09/55
```

You can standardize the separators (hyphens, blanks, and slashes) in the students' birthdates by defining your own collating sequence.

A directive file is used to sort the file BIRTHDATES. The SORT command to call the directive file is as follows:

```
SORT DIR=date_dir_file
```

The directive file DATE_DIR_FILE is shown below.

```
SORT, FROM=birthdates
SORT, KEY=((25, 2, mysequence))
SORT, KEY=((19, 3, mysequence))
SORT, KEY=((22, 3, mysequence))
SORT, SEQN=mysequence
SORT, SEQS=('0'..'9')
SORT, SEQS=('-', ' ', '/')
SORT, SEQA=YES
SORT, TO=dates_sorted
```

The SORT command defines a collating sequence named MYSEQUENCE. The first SEQS parameter specifies ten value steps from 0 through 9. This defines the order of the numbers. The next SEQS parameter specifies one step consisting of hyphens, blanks, and slashes. This defines the hyphen, blank, and slashes as equal values. The SEQA parameter specifies that blanks and slashes are to be output as hyphens. The file is sorted according to the date of birth.

The file DATES_SORTED output from the sort is shown below.

```
KING          M L 11-11-48
QUINTERA      L S 08-12-53
IRVING        W R 01-07-55
WALLACE       S T 12-09-55
EBERHARD      N I 06-05-58
ANDERSEN      C R 05-01-60
GREENWOOD     M R 09-12-61
ALLEN         M G 10-09-61
```

The file BIRTHDATES has been sorted in numeric order according to dates of birth, and the separators in the dates have been changed to hyphens in all records.

Part II: Keyed-File Utilities

Keyed-File Concepts 5

Keyed-File Concepts 5

The keyed-file utilities are a group of SCL utilities that can help you create and use keyed files. A keyed file is a file whose organization allows record access by key value. The keyed-file organizations are indexed-sequential and direct-access. This chapter describes keyed-file concepts so that you can more easily create and use keyed files.

You may already understand the concepts described in this chapter if you have used keyed files through a programming language (such as CYBIL, COBOL, or FORTRAN). Keyed-file concepts are also described in the CYBIL Keyed-File and Sort/Merge Interfaces manual, the COBOL Usage manual, and the FORTRAN Language Definition manual.

This chapter assumes that you already have a general understanding of NOS/VE files as described in the NOS/VE System Usage manual. It assumes that you have used sequential files and, possibly, byte-addressable files.

Sequential files, byte-addressable files and keyed files are alike in that all are written and read using record access. This means that the data in the files is contained in records.

A record is a collection of data that is read and written as a unit. A record could contain several fields of data, some of a fixed length and others varying in length. Thus, a record may have a fixed length or be variable in length.

For example, a record could contain three data items of different types: an integer, a floating point number, and a string of characters. To write a record, a program writes all three data items together as a record; when the record is later read, all three data items are delivered to the program.

The records in a sequential or byte-addressable file are stored as a simple sequence. The records in a keyed file are stored within a file structure.

Keyed-File Organizations

A keyed file is defined as such by its file_organization attribute. Currently, the keyed-file organizations are indexed-sequential and direct-access.

A keyed-file organization allows you to read any record in the file directly by specifying its key value. The key value for a record is determined when the record is written to the file.

The keyed-file interface performs all processing required to relate a key value to a record location; the user does not specify how this is done beyond choosing the file organization. The method of relating a key value to a record location differs for each keyed-file organization.

Indexed-Sequential File Organization

The indexed-sequential file organization allows content addressing of records; that is, you can directly access a record by the contents of one or more fields of data in the record. The fields of data by which a record is addressed are its key fields, and the contents of those fields are its key values.

An indexed-sequential file always has a primary key. (It can also have one or more alternate keys as described in the Alternate Keys section of this chapter.)

Each primary key value is unique within the file; there can be no duplicate primary-key values in a file.

The indexed-sequential file organization can be used only when you can assign a unique value to each record stored in the file. This unique value is usually a field of data within the record (an embedded key), although it can be a value assigned to the record and not included in the record data (a nonembedded key).

For example, the primary key for an employee file could be the employee's name. However, because two employees could have the same name, it is better to assign a unique identification number to each employee and use that number as the primary key for the file.

The indexed-sequential file organization should be used if a requirement exists to read file records both sequentially and randomly. For example, the records in an employee file could be read sequentially to produce a listing of all employees or read randomly to update individual records.

When an indexed-sequential file is read sequentially, its records are accessed in ascending order by key value. For example, if an employee file is read sequentially using its primary key (the employee identification number) the records are read in ascending order by their identification number. The order is kept even when new records are added to the file.

Indexed-Sequential File Structure

This section gives a general description of the indexed-sequential structure. You can use indexed-sequential files without knowing their structure. However, if you understand the indexed-sequential structure and how it grows, you can create more efficient indexed-sequential files by specifying appropriate values for structural parameters.

The internal structure of an indexed-sequential file is designed to provide both random and sequential access to the data records in the file. File space is divided into blocks, all the same size.

A block contains a block header and one of the following:

Internal tables
Data records (a data block)
Index records (an index block)

Each index record points to a data block. The index record contains the location of the data block and the range of key values of the data records stored in that block.

You can display the formatted contents of all components of an indexed-sequential file, the internal tables and index blocks as well as the data blocks, using the DISPLAY_KEYED_FILE command described in chapter 6.

As you might expect, the actual internal index mechanism is complex, but the simplified examples in this part provide the level of detail appropriate for indexed-sequential file use.

To see how an index works, let's look at a very small file that contains one index block and two data blocks. As shown in figure 5-1, the index block contains two index records. Each index record points to a data block in the file.

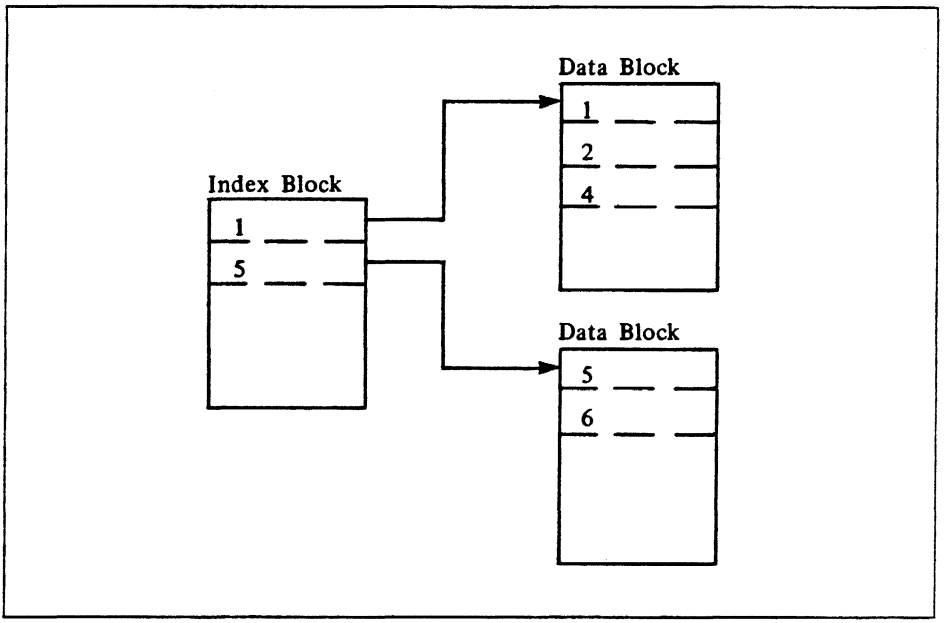


Figure 5-1. Minimal Indexed-Sequential Structure

Let's suppose you request to read randomly the record with key value 6. To read the record, these steps are performed:

1. The index records are searched to find the index record whose range of key values includes the key value 6.
2. After the correct index record (the second one) is found, the search for the record continues with the data block pointed to by the second index record.
3. The second data block is searched for the record with key value 6. When the record is found, its data is returned to the requester.

Next, suppose you request that all records in the file shown in figure 5-1 be read sequentially. These steps are performed.

1. The first index record is read to find the first data block.
2. The records from the first data block are read in order.
3. The second index record is read to find the second data block.
4. The records from the second data block are read in order.

5. The sequential read ends because there are no more index records, therefore, no more data blocks to read.

This process reads the records in key-value order because both the index records and the data records are kept in key-value order.

Data-Block Split

Usually, a block has some empty space, called padding, that was left empty so that additional records could be written later to the block. Suppose, as shown in figure 5-2, that a data block has been filled, a new record is to be written, and its key value is within the range of key values of the records in the full data block. For the file structure to be maintained, the data block must be split.

When a data-block split occurs, records in the data block whose key values are less than the key value of the new record remain in the existing block. All records in the existing block that come after the new record are moved to the newly created block.

The new record is put into either the new block or the existing block, depending on the relative amount of empty space in the blocks and the size of the new record. If the new record does not fit in either block, a second new block is created and the new record is put into that block.

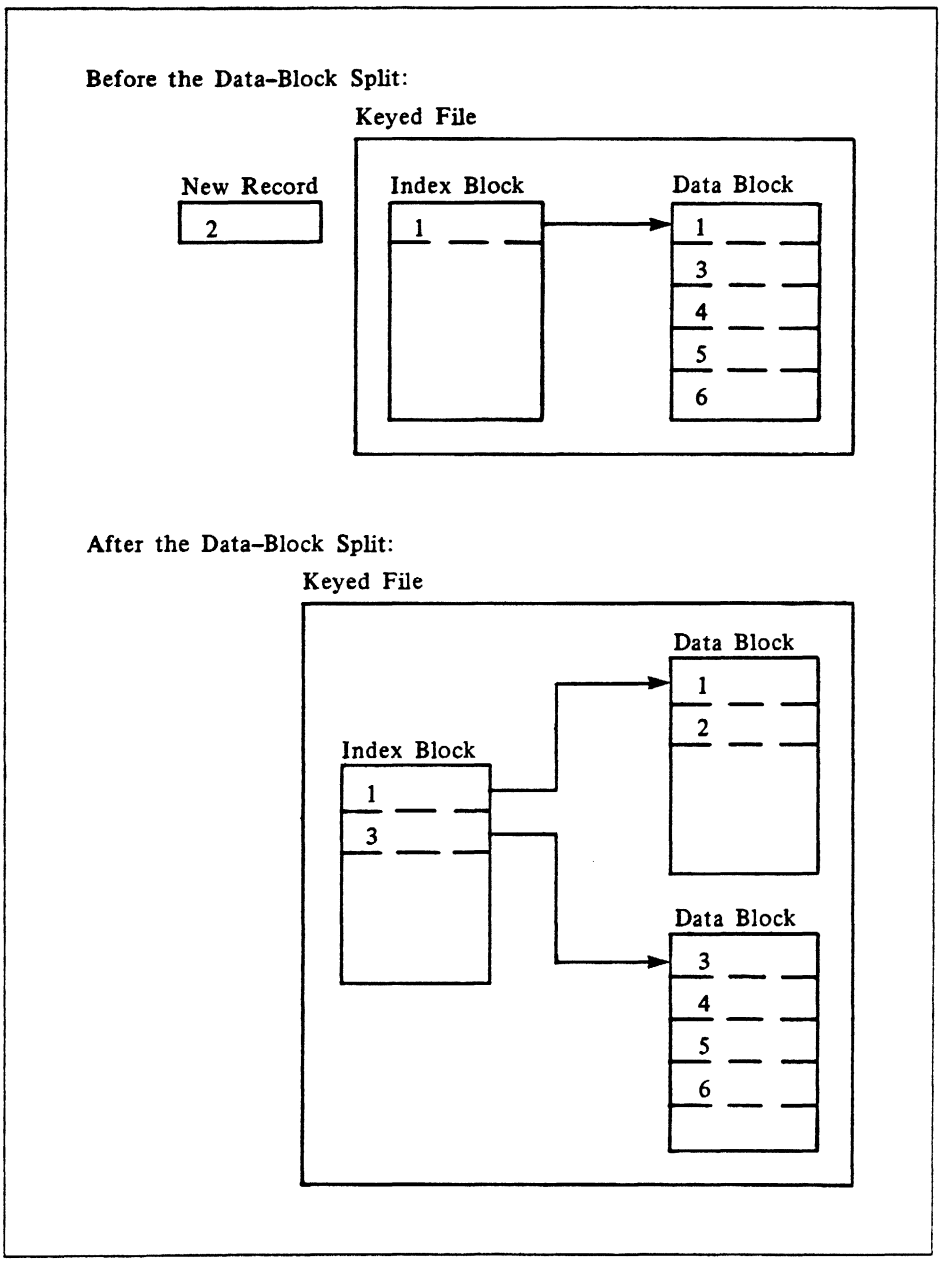


Figure 5-2. Data-Block Split

Index Levels

As with data blocks, index blocks are also initially created with some empty space (index-block padding). However, for each new data block created due to a data-block split, another index record must be created. With the addition of many data records, the initial index block becomes full. When the index block is full, the next data-block split causes an index-block split.

As shown in figure 5-3, when the initial index block splits, it causes the creation of another index level.

The index levels are numbered from the top down as index level 0, index level 1, and so forth. Index level 0 always has only one index block; it is always the starting point for an index search.

The index block at an upper level contains an index record for each index block at the next lower level. The index block at level 0 contains an index record for each index block at level 1.

A search for a data record requires an index-block search at each index level. The level-0 search finds the index record that points to the appropriate level-1 index block. If the file has only two index levels, the level 1 search finds the index record that points to the appropriate data block.

As you can see, the addition of another index level increases the time required to find an individual data record.

Index levels can be added up to the index-level limit of 15 levels. This sets a limit on the number of records in the file.

The index-level limit is reached when addition of another record to the file would require creation of another index level, but 15 index levels already exist in the file. When this happens, the index-level-overflow flag is set and no more records can be added to the file.

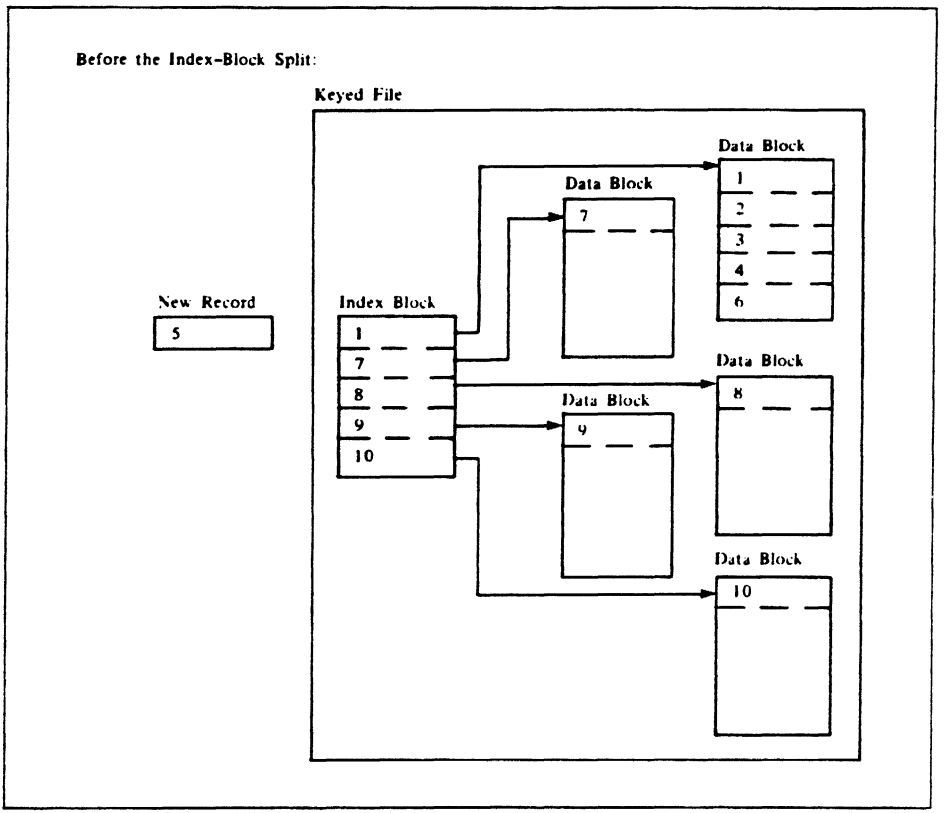


Figure 5-3. Index-Block Split

(Continued)

(Continued)

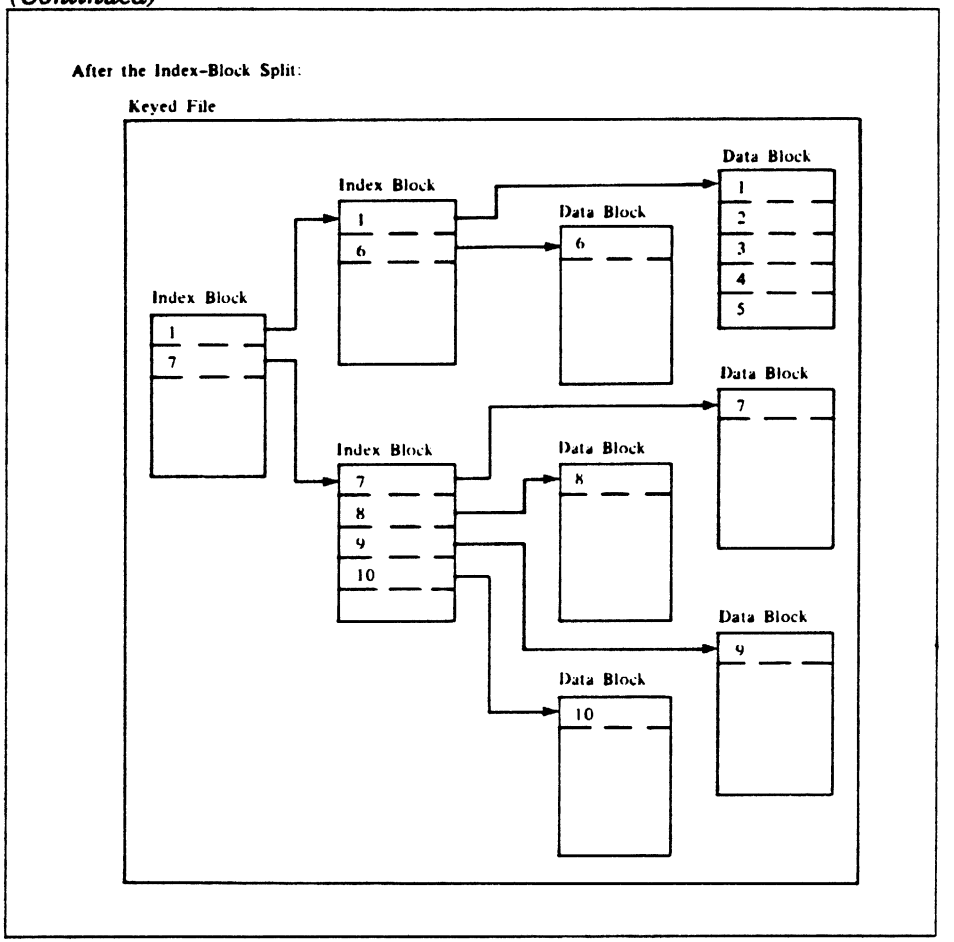


Figure 5-3. Index-Block Split

Indexed-Sequential Primary Keys

The primary key for an indexed-sequential file is defined when the file is created. The primary-key value must be unique for each record in the file.

A primary-key definition requires specification of these attributes:

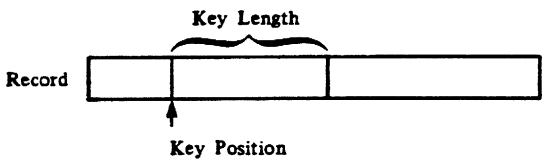
Embedded or nonembedded key (the default is embedded)
Key position (if the key is embedded)
Key length
Key type (the default is uncollated)

Collate-table name (if the key type is collated)

A key is embedded if the key value is part of the data in the record. An embedded key value is returned as part of the record data when the record is read; a nonembedded key value is not.

The key position in the record must be specified if the key is embedded. The first byte position in a record is byte 0. If the key is nonembedded, you do not specify a key position.

You must specify the key length whether the key is embedded or nonembedded. It indicates the number of bytes in the key.



The key type describes the data in the key. These are the possible key types:

Integer key	The key value is a signed binary value from 1 through 8 bytes long. (In general, except for packed CYBIL records, an integer value is written as 8 bytes.) Integer key values are sorted in ascending numerical order.
Uncollated key	The key value is a string of characters; it is sorted byte-by-byte according to the ASCII collating sequence.
Collated key	The key value is a string of characters; it is sorted byte-by-byte according to a collating sequence that you specify. If the key is a collated key, you must specify the collating sequence to be used to order the key values. The collating sequence is specified by its name. NOS/VE provides several predefined collating sequences (listed in appendix E). You can also create a collating sequence as described in appendix E.

Direct-Access File Organization

The second keyed-file organization is direct-access. In general, it should be used when fast individual record access is required, but sequential ordering of records using the primary key is not needed. (Sequential ordering is possible using an alternate key.) Also, the direct-access file organization is most effective when file updates (and the resulting file structure changes) are minimized. The following paragraphs compare the direct-access and indexed-sequential file organizations.

Both the indexed-sequential and direct-access file organizations use a primary key. You define the primary key for the file when you create the file. It can be a field embedded in the record or a nonembedded value. Each primary-key value in the file must be unique; the file can contain no duplicate primary-key values.

Like an indexed-sequential file, a direct-access file can have alternate keys. An alternate key for a direct-access file is the same as an alternate key for an indexed-sequential file. (See the Alternate Keys discussion later in this chapter.)

Like indexed-sequential file records, you must specify the primary-key value when writing or deleting a direct-access file record. Similarly, you must specify either a primary-key value or an alternate-key value to read a direct-access file record.

Direct-access and indexed-sequential files differ in their ordering of records in the file:

- When you read records sequentially from an indexed-sequential file, the records are returned in order, sorted by their primary-key values.
- A sequential pass through a direct-access file reads all records in the file, but the records are not returned in order by their primary-key values.

In general, random record access is faster for the direct-access file organization than for the indexed-sequential file organization. This is because the direct-access file organization determines the location of a record directly from its primary-key value, unlike indexed-sequential files where a record can be found only after a search at each index level.

Direct-Access File Structure

The direct-access file structure is designed to locate each record directly by its primary-key value. The primary-key value directly specifies the file block containing the record.

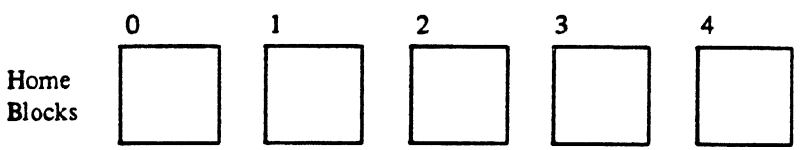
File space in a direct-access file is divided into equal-size blocks. Initially, all blocks in the file are home blocks (as opposed to overflow blocks).

When a record is written to a direct-access file, its primary-key value is hashed to produce the number of the home block in which the record is written. If the home block does not contain enough empty space for the new record, the record is written in an overflow block.

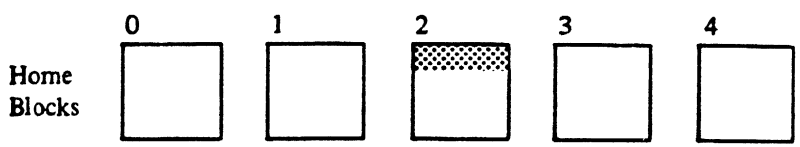
Assuming the hashing procedure produces a uniform distribution of numbers from the primary-key values in the file, the records are uniformly distributed among the home blocks of the file. Thus, each record can be found by a single search of its home block without additional searches of overflow blocks.

You specify the initial number of home blocks when you create the file. By default, a system hashing procedure is used to distribute the records among the home blocks although you can provide another hashing procedure for the file if you like.

As an illustration of a small direct access file, suppose you define a direct access file as having five home blocks.

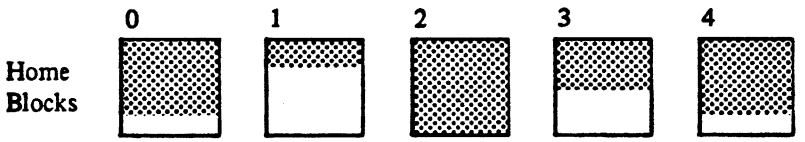


The first record written to the file has primary-key value XYZ. Assume that this primary-key value is hashed to produce the block number 2. The record is then written in home block 2.

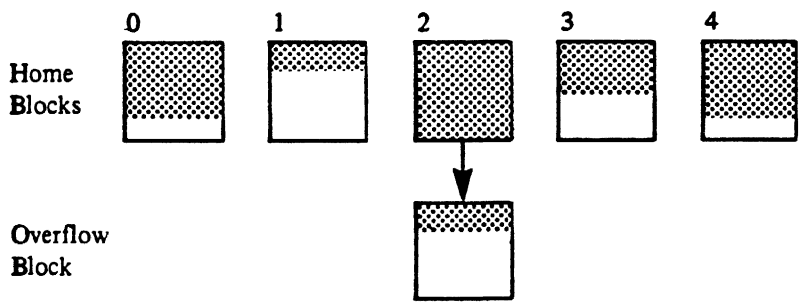


Assume you want to read the record with primary-key value XYZ. The value XYZ is hashed and, as before, produces the block number 2. The keyed-file interface searches for the record with primary-key value XYZ in home block 2. (The records in a block are ordered by primary-key value so each record can be quickly found.)

Suppose that many records have been written to the file and home block 2 has been filled.



At this point, a record is to be written with primary-key value ABC. Hashing of the value ABC produces block number 2, but there is insufficient space for the record in home block 2 so it is written in an overflow block.



Later, to read the record with primary-key value ABC, the primary-key value is hashed to produce block number 2. Home block 2 is searched for primary-key value ABC. When it is not found in the home block, the search continues in the overflow block until the record is found.

An ideal direct-access file structure has these characteristics:

- Sufficient home blocks are allocated and records are uniformly distributed among the home blocks so as to avoid overflow.
- Each block contains a limited number of records so as to minimize the search time in each block.

- The number of home blocks is not so large that the file contains excessive unused space.

These characteristics are determined by the file attribute values specified when the file is created. You must specify the initial_home_block_count and can optionally specify the maximum_block_length and hashing_procedure_name attributes. (The attribute parameters are described in chapter 6.)

One other characteristic to be considered when selecting the number of home blocks is the loading factor. The loading factor is the percentage of block space used. To allow for less-than-uniform distribution of records in the home blocks, the loading factor should be no greater than 90%.

You can use the following equations to determine the minimum home_block_count for a given loading factor if the number of bytes of data in the file and the block size are known.

If the file has fixed-length records, reduce the block size by 39 bytes, as follows:

$$\text{home\_block\_count} = \frac{\text{record\_count} \times \text{fixed\_record\_length}}{(\text{loading\_factor}/100) \times (\text{block\_size} - 39)}$$

If the file has variable-length records, reduce the block size by 36 bytes and use the average record length plus 3 as the record length, as follows:

$$\text{home\_block\_count} = \frac{\text{record\_count} \times (\text{average\_record\_length} + 3)}{(\text{loading\_factor}/100) \times (\text{block\_size} - 36)}$$

To illustrate, suppose the direct-access file is to contain 10,000 80-byte records (80,000 bytes of record data). Using a block size of 4096 bytes and a loading factor of 90%, the equation appears as follows:

$$\text{home\_block\_count} = \frac{10000 \times 80}{(90/100) \times (4096 - 39)}$$

The equation gives 22 blocks as the minimum home block count for the file. However, it is recommended that the home block count be a prime number so 23 would be a better home block count for the file in this example.

Hashing Procedures

The system provides a default hashing procedure named AMP$SYSTEM_HASHING_PROCEDURE or, you may specify your own hashing procedure. This would be appropriate if the procedure would produce a more uniform distribution of numbers from the primary-key values in your file.

The system executes the hashing procedure each time a record is requested by key value from the direct-access file. The hashing procedure is not stored with the file so the system must be able to load the procedure each time the direct-access file is opened.

Hashing procedures can only be written in the CYBIL programming language. For more information on writing a hashing procedure, see the CYBIL Keyed-File and Sort/Merge Interfaces manual.

Direct-Access Primary Keys

In general, the primary key of a direct-access file has the same characteristics as the primary key of an indexed-sequential file. You specify whether the primary key is embedded or nonembedded, its position (if the key is embedded), and the key length. However, for direct-access files, the specified KEY_TYPE attribute value is ignored; the KEY_TYPE attribute for a direct-access file is always uncollated.

Unlike an indexed-sequential file, sequential access calls to a direct-access file while the primary key is selected do not return the file records sorted by primary-key value. The calls return records according to their physical location in the direct-access file. Records within a block are ordered according to the default ASCII collating sequence, but the blocks are not ordered by primary-key values.

Direct-access file records can be accessed in order if one or more alternate keys are defined for the file. The alternate index keeps the alternate-key values in sorted order. Sequential access calls while an alternate key is selected return records in the order provided by the alternate index.

If appropriate, you could define an alternate key for the same field as an embedded primary key. In this way, you could access direct-access file records in primary-key value order.

NOTE

If you specify a collation table for a direct-access file using the COLLATE_TABLE_NAME attribute, the collation table is loaded when the new file is first opened. However, the collation table is not used by the primary key, nor can it be used by any alternate key.

Alternate Keys

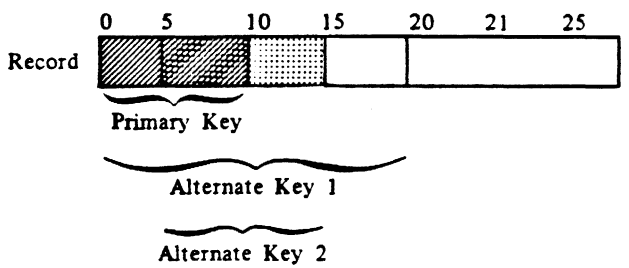
A record within a keyed file can always be accessed by its primary-key value. An alternate key provides an additional way to access records.

An alternate key defines a value in the data record by which the record can be accessed. An alternate key is defined as a field or group of fields in the record.

Although a program can use alternate keys to read records or to position a file, alternate keys cannot be used to write, replace, or delete records. The primary-key value must be used to identify a record to be written, replaced, or deleted.

Alternate-Key Characteristics

Alternate-key fields can overlap each other and the primary key. For example, the primary-key field could be bytes 0 through 9 and two alternate-key fields bytes 0 through 19 and bytes 4 through 14.



Unlike a primary-key value, one alternate-key value can be associated with several records in a file. This is because an alternate-key value need not be unique. The same alternate-key value can occur in several records. For example, the same job title can be associated with many names as follows:

Data Record:

```
Hanson    Computer Programmer
Jones     Computer Programmer
Smith     Computer Programmer
```

Alternate Index:	Alternate-Key Value	Primary-Key Value
	Computer Programmer	Hanson
		Jones
		Smith

A record can contain more than one alternate-key value if the alternate key is defined as a field that repeats in the record; thus, a single record could contain several alternate-key values. For example, the license numbers of several cars owned by one person as follows:

Data Record: R. Petty 1 LB AU 2ASM451 ELK 592

Alternate Index:	Alternate-Key Value	Primary-Key Value
	1 LB AU	R. Petty
	2ASM451R	R. Petty
	ELK 592	R. Petty

The Alternate Index

The index for the primary key was described earlier in this chapter. Each alternate key defined for a file has its own index.

An alternate index contains index records, each of which associates an alternate-key value with the primary-key values of the records containing that alternate-key value. The list of primary-key values associated with an alternate-key value is the key list for that alternate-key value.

When you select an alternate key and then specify an alternate-key value, the system searches for the value in the alternate index. If it finds the alternate-key value, it uses the primary-key values in the key list for the alternate-key value to access the data records.

When one or more alternate keys are defined for a file, file updates require more time because the alternate indexes must also be updated. Alternate keys should be used only when the additional record access capability offsets the cost of increased time spent for file updates.

Alternate-Key Definition

The attributes of an alternate key are specified by its alternate-key definition.

These attributes are required to define an alternate-key field:

Key name
Key position
Key length

An alternate key has a name so that it can be selected for use. The alternate-key position and length define the alternate-key field within the record.

These optional attributes define how the alternate key is processed:

Key type
Collate table name (if the key type is collated)
Duplicate key values
Null suppression
Sparse-key control
Repeating groups
Concatenated key
Variable-length key

The key type of an alternate key determines the order of the alternate-key values in the alternate index, and therefore, the order in which records are accessed sequentially when you use the alternate key. The key types for an alternate key are the same as the key types for the primary key as described earlier in this chapter.

Collated Alternate Key

If the key type is collated, the alternate key requires a collation table. In most cases, you should explicitly specify the collation table to be used. However, if the file is an indexed-sequential file with a collated primary key, you can use the primary-key collation table as the default collation table for the alternate key.

Collated key values are stored in collated form in the index. The collation is performed after the key values are read from the file and immediately before the values are stored in the index.

Thus, collation does not affect the selection of key values for the index. When null suppression is used, the key value is determined to be a null value before collation. Similarly, when the key is a variable-length key, collation does not apply to the key delimiter characters.

Duplicate Key Values

By default, duplicate values for an alternate key are not allowed. However, if you want to allow duplicate key values, you can specify whether the records having the same alternate-key value are accessed in primary-key-value order or in first-in, first-out order.

In a key list ordered by primary key, the primary-key values are stored in sorted order according to the primary-key type. New values are added to the key list so that the primary-key-value order is kept.

In a key list ordered first-in, first-out, the primary-key values are stored in the key list in the order the values are added to the key list, instead of in primary-key-value order. New values are always added to the end of the list.

For Better Performance

When alternate-key values are frequently duplicated in a file, the key lists should be ordered by primary-key value. First-in, first-out ordering of key lists requires that delete and replace operations sequentially search the key list to find the primary-key value; a sorted key list provides faster access to a primary-key value.

For example, suppose you write three records to the file in this order:

```
McDarrels     Hamburgers
Burger Duke   Hamburgers
Willys        Hamburgers
```

The following shows the resulting key list in primary-key order and in first-in, first-out order:

Alternate Key Value	Key List - Ordered by Primary Key	Key List - First-In, First-Out
Hamburgers	Burger Duke	McDarrels
	McDarrels	Burger Duke
	Willys	Willys

Duplicate Key Value Error Processing

If duplicate values are not allowed and a duplicate is found in a record about to be written to the file, the record is not written to the file and a nonfatal error (aae$duplicate_alternate_key) is returned.

A nonfatal error (aae$unexpected_dup_encountered) also occurs if a duplicate value is found while a new alternate index is being created. However, the record containing the duplicate value cannot be discarded, as it is already in the file. Subsequent processing depends on whether incrementing the nonfatal-error count causes the count to reach the nonfatal-error limit as set by the user.

- If the nonfatal-error limit is not reached, the alternate key being applied is redefined to allow duplicates, ordered by primary-key value, and the current apply operation continues.
- If the nonfatal-error limit is reached, the error condition aae$duplicate_key_limit occurs and the effects of the current apply operation are undone as far as possible. Deletions cannot be undone, but any creations that have taken place are undone, and a message is issued for each one.

In either case, a message describing the action taken is written to the $ERRORS file.

Null Suppression

By default, if an alternate-key field contains a null value, the null value is stored as the alternate-key value for the record. The null_suppression attribute allows you to exclude null values from an alternate index.

Null suppression excludes any record with a null alternate-key value from the alternate index. Null suppression can save space, access time, and update time because the index is smaller when null alternate-key values are excluded. (Null suppression does not remove the null value from the data record.)

The null value depends on the key type as follows:

Key Type	Null Value
Integer	Zero
Uncollated	Spaces
Collated	Spaces (before collation)

If null suppression is not specified, records containing a null value in the alternate-key field are indexed by the null value. The records can later be accessed by specifying the null value as the alternate-key value.

For example, suppose the spouse's name is defined as an alternate key to a membership file. Unmarried members would have a null value for the alternate-key field. Therefore, the key list for the null value lists all unmarried members. The following shows the alternate index with and without null suppression:

Without Null Suppression		With Null Suppression	
Spouse's Name	**Member's ID**	**Spouse's Name**	**Member's ID**
	1626736	Diana Simmons	4872672
	8273648	Mark Ramsey	2673651
Diana Simmons	4872672	Shelly Gable	7726184
Mark Ramsey	7726184		
Shelly Gable	2673651		

Sparse-Key Control

You can use sparse-key control to create an alternate index that includes or excludes records depending on the character in a specific position in the record (the sparse-key control position).

The sparse-key control position must be within the minimum record length. If you specify sparse-key control for an alternate key, the alternate-key field or fields need not be within the minimum record length.

If the character at the sparse-key control position indicates that the record should be included in the alternate index, but the record has no alternate-key value because the record ends before the alternate-key field, the record is not included in the alternate index. Although the record is not included in the alternate index, it is written to the file and a trivial error (AAE$SPARSE_KEY_BEYOND_EOR) is returned.

For example, suppose a student file has a one-character code indicating the student's class. To get a mailing list for juniors and seniors only, you could define an alternate index controlled by the class code.

To specify sparse-key control, you specify three values:

Value	Example
Sparse-key control position	Position of the class code in the record
Sparse-key control characters	Junior and senior class code characters
Sparse-key control effect (Indicates whether the alternate-key value should be included or excluded if the sparse-key character matches)	Included if the class code indicates a junior or senior record

Assume that the sparse-key control position is the first character after the name field and that the junior and senior class codes are 3 and 4. If the following records are copied to the file, the first three records are included in the alternate index, but not the last record.

```
Louis Skolnik     4
Gilbert Sullivan  4
Elliot Wermzer    3
Judy Manhasset    2
```

Concatenated Keys

A concatenated key is an alternate key formed from several fields, or pieces, in the record. A concatenated key can comprise up to 64 pieces.

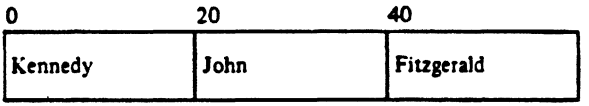
The concatenated pieces can be noncontiguous and can be concatenated in any order. Each piece can be a different key type. All collated-key pieces use the same collation table.

The first piece you specify is the leftmost piece of the key. You specify it the same as you specify a nonconcatenated key. The pieces to be concatenated to the leftmost field are defined by individual subcommands. The subcommand order specifies the order of the concatenated pieces.

A concatenated key can use sparse-key control and/or null suppression. A concatenated key is considered to have a null value if the values in all fields of the key are null (before collation for collated keys).

For example, suppose you decide to define an alternate key consisting of the initials of the member's name. The first piece of the key value would be the first letter of the member's first name, the second piece

would be the first letter of the member's middle name, and the third piece would be the first letter of the member's last name. Consider this data record:



The alternate key value is JFK, assuming the concatenated-key pieces are defined as:

First piece:	`Key_Position=20, Key_Length=1`
Second piece:	`Key_Position=40, Key_Length=1`
Third piece:	`Key_Position=0, Key_Length=1`

Repeating Groups

The repeating-groups attribute allows a data record to contain more than one value for the same alternate key. This allows a primary-key value to be associated with more than one alternate-key value.

To specify an alternate-key field within a repeating group:

1. Specify the first alternate-key field by its key position, key length, and key type. All subsequent alternate-key fields have the same length and type as the first.
2. Specify repeating groups for the alternate key by specifying the repeating group length, that is, the distance from the beginning of the first instance of the alternate key to the beginning of the second instance of the alternate key in the record.
3. Specify the repeating-group count, that is, how many times the alternate key field repeats in the record.

You can specify that the repeating group repeats a fixed number of times or that it repeats until the end of the record.

- If the alternate-key field repeats a fixed number of times, all alternate-key fields must be within the minimum record length.

- If the alternate-key field repeats to the end of the record, the minimum record length imposes no restriction. The system stores as many alternate-key values as the record length allows; it ignores trailing information not long enough to contain an alternate-key value.

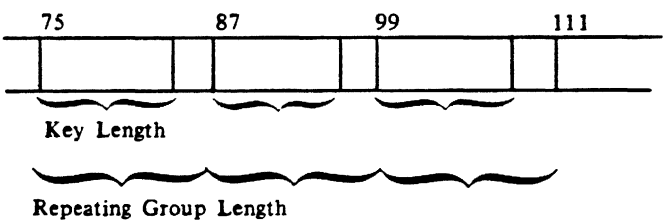
Repeating groups cannot be used with concatenated keys or when duplicate-key values are allowed and ordered first-in-first-out.

For example, suppose each record in a membership file lists the sports the member enjoys and his years of experience as follows (columns are counted from zero):

Field: Sports and Sports Experience

Columns: Variable number of 2-field pairs beginning at column 75 The Sports field is 10 characters followed by a 2-digit Sports Experience field

Type: ASCII characters



You could define an alternate key for the Sports values (without the Sports-Experience values) as follows:

```
Key_Position=75, Key_Length=10, Key_Type=uncollated,
Repeating_Group_Length=12,
Repeating_Group_Count=repeat_to_end_record,
Duplicate_Key_Values=ordered_by_primary_key
```

The key list for an alternate-key value would list the identification numbers of all members that enjoy that sport.

The following shows the primary keys for three records and their contents from column 75 to the end of the record:

Primary Key	Record Contents Beginning at Column 75
1662876	Volleyball02Running 03Basketball02
6166287	Bicycling 10Volleyball01
0027840	Running 15Running 15Running 15

If these were the only records in the file, the alternate index would appear as follows:

Alternate-Key Value	Primary-Key Value
Basketball	1662876
Bicycling	6166287
Running	0027840 1662876
Volleyball	1662876 6166287

Notice that because the key type is Uncollated and the duplicate-key values specification is Ordered_by_primary_key, each key list is sorted according to the default ASCII collating sequence.

Notice also, as shown by the Running key list, each primary-key value is listed only once in a key list, regardless of the number of times the alternate-key value occurs in the record.

Variable-Length Key

A variable-length alternate key is an alternate key whose values vary in length. Its alternate-key definition specifies its starting position, its maximum length, and its set of delimiter characters. The end of a variable-length key value is marked by a delimiter character, the end of the key field, or the end of the record, whichever is found first starting at the key_position.

By defining the key as a variable-length key, you can use the following values as alternate keys:

- The first value beginning at a certain position of the record.
- The last field in a variable-length record.
- All data in a variable-length record.

By defining the key as a variable-length key with the repeating groups attribute, you can use these values as alternate keys:

- A value found anywhere in a fixed-length field (if all other characters in the field are in the set of delimiter characters for the alternate key).

- A value found anywhere in a fixed-length field (if all other characters in the field are in the set of delimiter characters for the alternate key).

- Each value in a sequence of values, separated by one or more consecutive delimiter characters. The sequence of values can be within:

  - A fixed-length field.

  - A variable-length field at the end of the record.

  - The entire record.

For Better Performance

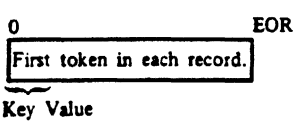
Define a key as a variable-length key only when necessary. The requirement to scan the key field for delimiter characters adds processing time when the alternate index is built and when the file is updated.

The following examples use the CREATE_ALTERNATE_INDEXES subcommand CREATE_KEY_DEFINITION to define variable-length keys.

Example 1:

This subcommand defines the first sequence of non-blank characters in each record as an alternate-key value. The maximum key-value length is 80 characters.

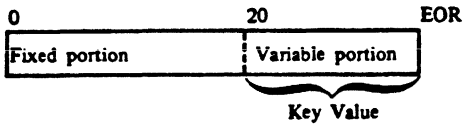
```
create_key_definition, key_name=first_token, ..
  key_position=0, key_length=80, variable_length_key=' '
```



If the entire record is 80 characters or less and the record contains no blanks, the key value would be the entire record.

Example 2:

Suppose each record consists of a required 20-byte portion and an optional variable-length portion of up to 120 bytes.



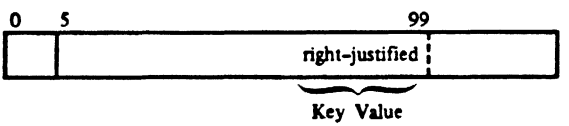
This subcommand defines the variable-length portion as an alternate key.

```
create_key_definition, key_name=variable_portion, ..
  key_position=20, key_length=120, variable_length_key=''
```

The null string ('') defines an empty delimiter set, indicating that the end of the key value is marked by either the end of the 120-byte field or the end of the record.

Example 3:

Suppose a 100-byte field at byte 5 contains one value from 0 through 100 bytes, right-justified and blank-filled.



This subcommand defines the value as a variable-length key.

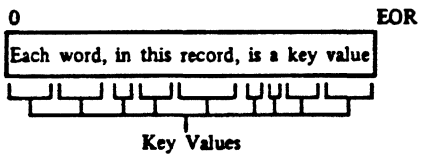
```
create_key_definition, key_name=right_just, ..
  key_position=5, key_length=100, ..
  variable_length_key=' ', ..
  repeating_group_length=1, repeating_group_count=256
```

Because the value is right-justified in the field, the key must be defined with the repeating groups attribute so the search for the value does not end at the first delimiter.

For a repeating variable-length key, the repeating_group_length value can be any integer greater than zero. (The actual value is irrelevant.) The repeating_group_count is the length of the alternate-key field.

Example 4:

Suppose that each token in a record is to be a key value.



To define each string of letters in the data as a key value, first, define an SCL string variable containing all ASCII characters except the uppercase and lowercase letters, then define the key using the string variable.

```
create_variable, key_delimiters, kind=(string,76),..
  value=' 1234567890-=!@#$%^&*()_+[]`{}~;'' :"|,./<>?'..
//$CHAR(000)//$CHAR(001)//$CHAR(002)//$CHAR(003)..
//$CHAR(004)//$CHAR(005)//$CHAR(006)//$CHAR(007)..
//$CHAR(008)//$CHAR(009)//$CHAR(010)//$CHAR(011)..
//$CHAR(012)//$CHAR(013)//$CHAR(014)//$CHAR(015)..
//$CHAR(016)//$CHAR(017)//$CHAR(018)//$CHAR(019)..
//$CHAR(020)//$CHAR(021)//$CHAR(022)//$CHAR(023)..
//$CHAR(024)//$CHAR(025)//$CHAR(026)//$CHAR(027)..
//$CHAR(028)//$CHAR(029)//$CHAR(030)//$CHAR(031)..
//$CHAR(127)
```

Notice that the concatenation operator (//) must be left-justified on the line so that no extra spaces are put in the string. No spaces can precede the continuation (..) indicator.

This command defines the key using the string variable:

```
create_key_definition, key_name=words,..
  key_position=0, key_length=80,..
  variable_length_key=key_delimiters,..
  repeating_group_length=1,..
  repeating_group_count=repeat_to_end_of_record
```

The repeating_group_length can be any integer greater than zero. (The actual value is irrelevant). The repeating_group_count is the alternate-key field length; repeat_to_end_of_record specifies that the sequence of values continues until the end of the record.

Nested Files

A nested file is a keyed-file structure defined within a NOS/VE file cycle. It is recognized and used by the keyed-file interface only; it is not recognized or used by the NOS/VE file system.

All nested files in a file share the same catalog entry so if one nested file is damaged and cannot be accessed, all nested files in the file are considered damaged and cannot be accessed.

The keyed-file interface provides nested files so as to extend the NOS/VE limit on the number of files a task can use. All nested files defined in a file share the same memory segment. This provides effective memory use when the nested files are much smaller than the segment size limit (2^{32} bytes).

The keyed-file interface creates the initial nested file (named $MAIN_FILE) when it creates the keyed file. The nested file $MAIN_FILE is always the default nested file used when no other nested file is explicitly selected.

Currently, additional nested files can be created by:

- A CYBIL program (as described in the CYBIL Keyed-File and Sort/Merge Interfaces manual)
- A COPY_KEYED_FILE command that copies an existing nested file
- The CREATE NESTED FILE subcommand of the CREATE_KEYED_FILE and CHANGE_KEYED_FILE utilities

When creating a nested file, COPY_KEYED_FILE uses the attributes of the nested file copied. The CREATE_NESTED_FILE subcommand defines the attributes of the nested file created.

The following attributes belong to each nested file individually:

- File organization (indexed-sequential or direct-access)
- Record attributes, including the record type and the minimum and maximum record lengths
- Primary-key attributes, including its key position, key length, key type, and collation table
- Compression procedure name

- Structural attributes applicable to the file organization

The display produced by a DISPLAY_KEYED_FILE_PROPERTIES command lists the attributes of each nested file separately (as described in chapter 6).

All other file attributes apply to all nested files in a keyed file. The RECORD_LIMIT attribute specifies the maximum number of records in each nested file.

Each alternate-key definition applies to only one nested file, the nested file selected when the alternate key is defined.

Displaying, Copying, and Creating Keyed Files 6

Displaying, Copying, and Creating Keyed Files 6

This chapter describes several basic tasks you can perform on keyed files using SCL commands.

Task	Commands
Display the properties of an existing keyed file	DISPLAY_KEYED_FILE_PROPERTIES
Dump the internal structure and contents of an existing keyed file	DISPLAY_KEYED_FILE
Copy a file record-by-record	COPY_KEYED_FILE
Create a keyed file and copy data to the new file	SET_FILE_ATTRIBUTES and COPY_KEYED_FILE
Re-create an existing keyed file to improve the file structure	SET_FILE_ATTRIBUTES and COPY_KEYED_FILE

It is assumed that the keyed-file tasks described in this part are used to prepare files for programs that read keyed files. For information on writing programs that use keyed files, refer to the CYBIL Keyed-File and Sort/Merge Interfaces manual, the FORTRAN Language Definition manual, or the COBOL Usage manual.

Keyed-File Displays

To list the contents and properties of an existing keyed file, use the keyed-file display commands DISPLAY_KEYED_FILE and DISPLAY_KEYED_FILE_PROPERTIES.

The DISPLAY_KEYED_FILE_PROPERTIES command can list file attributes and statistics for an existing keyed file. If the file has alternate keys, it also lists the alternate-key attributes.

The DISPLAY_KEYED_FILE_PROPERTIES display can indicate a structural error condition by the contents of the Altered_Not_Closed and Ruined_Flag fields. The other structural properties displayed show the efficiency of the file structure.

When a structural problem appears in a keyed file, the DISPLAY_KEYED_FILE command can be used to produce a formatted dump of the part of the keyed file that is in error. A dump of the file may help recover the data and fix any software error that caused the structural error in the file.

DISPLAY_KEYED_FILE_PROPERTIES Command

Purpose Displays properties of a keyed file. The displayed properties can include file attributes, structural properties, and statistics.

Format

DISPLAY_KEYED_FILE_PROPERTIES (DISKFP)
FILE=file or (file, nested_file_name)
OUTPUT=file
DISPLAY_OPTIONS=list of keyword_value
STATUS=status_variable

Parameters

FILE or F

Keyed file for which properties are to be displayed. You must have at least read permission to the file. This parameter is required.

To specify a nested file, first specify the file reference and then the nested-file name, enclosed in parentheses.

OUTPUT or *O*

File to which the display is written. If you omit the OUTPUT parameter, the display is written to file $OUTPUT.

DISPLAY_OPTIONS or *DISPLAY_OPTION or DO*

List of one or more keyword values indicating the property types to be displayed.

FILE_ATTRIBUTES or FA	File attributes kept for the life of the keyed file.
STATISTICS or S	Statistics maintained for the keyed file.
STRUCTURAL_ PROPERTIES or SP	Internal organization properties of the keyed file.
ALL or A	All of the above. (You cannot specify other keywords with ALL.)

If you omit the DISPLAY_OPTIONS parameter, the display includes the file attributes and structural properties, but not statistics.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- The display consists of two or more pages of output.
  - The first page lists the properties that pertain to the entire file.
  - The second and any subsequent pages list the properties of each nested file in the file and the alternate keys defined for each nested file.

  Unless additional nested files have been created in the file, a keyed file contains only one nested file; it is named $MAIN_FILE.

- At the file level, the file attributes and their possible values are as follows:

Application_Information	:	none or string
Block_Length "actual"	:	4096
Error_Exit_Name	:	none or name
File_Access_Procedure	:	none or name
File_Content	:	UNKNOWN, LIST, LEGIBLE, OBJECT, or SOURCE
File_Limit	:	integer "bytes"
Forced_Write	:	unforced, forced or forced_if_structure_change
Log_Residence	:	none or catalog path
Logging_Options	:	none, enable_media_recovery, enable_request_recovery, or all
Maximum_Record_Length	:	integer "bytes"
Minimum_Record_Length	:	integer "bytes"
Open_Position	:	$boi, $bop, $eoi or $asis

Permanent	:	yes or no
Record_Limit	:	integer
Ring_Attributes	:	(integer, integer, integer)
Size	:	integer "blocks"
User_Information	:	none or string

This page intentionally left blank.

Nested File Attributes

- For each nested file, the file attributes and their possible values are as follows:

Compression_Procedure_Name	:	name
Creation_Date	:	mm/dd/yy hh:mm:ss.nnn
Embedded_Key	:	yes or no
File_Organization	:	indexed_sequential or direct_access
Key_Length	:	integer "bytes"
Key_Position	:	integer
Key_Type	:	uncollated, collated or integer
Maximum_Record_Length	:	integer "bytes"
Minimum_Record_Length	:	integer "bytes"
Record_Type	:	undefined, variable or fixed

The key values apply only to the primary key.

In addition, for indexed-sequential files only:

Collate_Table_Name	:	name
Character_Mapping	:	$CHAR(character_ordinal) "symbol" => collating weight in hex.
Character_Ordering	:	collate sequence position => character
Data_Padding	:	integer "%"
Index_Padding	:	integer "%"

In addition, for direct-access files only:

Dynamic_Home_Block_Space	:	yes or no
Hashing_Procedure_Name	:	name or AMP$SYSTEM_HASHING_PROCEDURE
Home_Block_Count	:	integer

Alternate-Key Attributes

- For each alternate key, DISPLAY_KEYED_FILE_PROPERTIES lists only those properties defined for the key. These are the possible alternate-key file attributes and values:

Collate_Table_Name	:	name or (defaulted to primary-key table)
Character_Mapping	:	$CHAR(character_ordinal) "symbol" => collating weight in hex.
Character_Ordering	:	collate sequence position => character
Concatenated_Key	:	yes or no
Key_Length "Piece nn"	:	integer "bytes"
Key_Position "Piece nn"	:	integer
Key_Type "Piece nn"	:	uncollated, collated or integer
Creation_Date	:	mm/dd/yy hh:mm:ss.nnn
Duplicate_Key_Values	:	ordered_by_primary_key, first_in_first_out, or not_allowed
Key_Group_Name	:	name
Null_Suppression	:	yes or no
Repeating_Groups_Specified	:	yes or no
Repeating_Group_Count	:	integer
Repeating_Group_Length	:	integer "bytes"
Sparse_Key_Control	:	yes or no
Sparse_Key_Control_Position	:	integer
Sparse_Key_Control_Characters	:	$CHAR(nnn) "x" // $CHAR(nnn) "x"
Sparse_Key_Control_Effect	:	include_key_value or exclude_key_value
Variable_Length_Key	:	yes
Key_Delimiter_Characters	:	$CHAR(nnn) "x" // $CHAR(nnn) "x"

- The values subordinate to the Concatenated_Key, Repeating_Groups_Specified, Sparse_Keys, and Variable_Length_Key fields are displayed only if defined for the key.

  If the key is not a concatenated key, only one set of Key_Length, Key_Position, and Key_Type values is listed. Otherwise, a set is listed for each piece of the concatenated key.

- The Collate_Table_Name file attribute is displayed only if a collated key is defined for the file (either the primary key or an alternate key). The Collate_Table itself is displayed if the collation table name does not begin with OSV$ (that is, the collating sequence is not one of the NOS/VE collating sequences listed in appendix E).

  The collate table is displayed twice: The first display shows the collating weight assigned to each character, and the second display shows the characters in weight order.

- At the file level, the structural properties and their possible values are as follows:

Altered_Not_Closed	:	yes or no
Nested_File_Count	:	integer
Ruined_Flag	:	off, file_ruined_at_flush, bad_rasp_structure, read_error, write_error, alternate_key_mismatch, cannot_find_rasp, bad_empty_chain
Segment_Information		
Blocks_In_Use	:	integer
Empty_Block_Count	:	integer

- At the nested-file level, the structural properties and their possible values are as follows:

  Block_Count : integer
  Ruined_Flag : (same as at file level)

  In addition, for indexed-sequential files only:

  Index_Levels "current" : integer
  Index_Level_Overflow : yes or no

- For each alternate key, the structural properties and their possible values are as follows:

  Block_Count : integer
  Index_Levels "current" : integer
  Index_Level_Overflow : yes or no
  Ruined_Flag : (same as at file level)

- You can use DISPLAY_KEYED_FILE_PROPERTIES to determine whether the keyed-file structure is intact.

  - The structural property Altered_Not_Closed is a flag that indicates a structural error. It is most often set when a system failure during file modification prevents the file from being closed (flushed).

  - The structural property Ruined_Flag lists the structural error condition if one exists. If the Ruined_Flag is set for any nested file or alternate key in a file, it is set at the file level also.

- This statistic is listed at the file level:

  Segment_Information
  Last_Flush : mm/dd/yy hh:mm:ss.nnn

- These statistics are listed at the nested file level:

Alternate_Keys	:	integer
File_Accesses		
Open_Count	:	integer
Delete_Count	:	integer
Get_Count	:	integer
Get_Next_Count	:	integer
Put_Count	:	integer
Putrep_Count	:	integer
Replace_Count	:	integer
Record_Count	:	integer

- In addition, for direct-access files only:

Overflow_Block_ Count	:	integer
Overflow_Record_ Count	:	integer

- The file access statistics listed may be inaccurate if the file has been read without modify permission. The reason for this is that when the file is read without modify permission, the statistics for that read cannot be recorded.

- If the file specified on the command is not a keyed file, DISPLAY_KEYED_FILE_PROPERTIES returns the warning status, AAE$FILE_IS_NOT_A_KEYED_FILE.

- If the file specified on the command does not exist, DISPLAY_KEYED_FILE_PROPERTIES returns the warning status, AAE$FILE_DOES_NOT_EXIST.

Examples This command lists statistics and structural properties for file $USER.KEYED_FILE on file $USER.LIST:

```
display_keyed_file_properties ..
file=$user.keyed_file output=$user.list ..
display_option=(statistics, structural_properties)
```

This command lists the file attributes and structural properties of file $USER.ISFIL on $OUTPUT. The resulting display is shown:

```
/diskfp $user.isfil
                Display Keyed_File_Properties          1986-11-03
NOS/VE Keyed File Utilities 1.3 85259                    10:31:23
File = .NVE.USER99.ISFIL

File_attributes and structural_properties at the
 file level
   Altered_Not_Closed       : no
   Application_Information  : none
   Block_Length "actual"    : 4096 "bytes"
   Error_Exit_Name          : none
   File_Access_Procedure    : none
   File_Content             : UNKNOWN
   File_Limit               : 4398046511103 "bytes"
   Forced_Write             : unforced
   Log_Residence            : none
   Logging_Options          : none
   Maximum_Record_Length    : 80 "bytes"
   Minimum_Record_Length    : 50 "bytes"
   Nested_File_Count        : 1
   Open_Position            : $boi
   Permanent                : yes
   Record_Limit             : 4398046511103
   Ring_Attributes          : (11, 11, 11)
   Ruined_Flag              : off
   Segment_Information
      Blocks_In_Use         : 2
      Empty_Block_Count     : 0
   Size                     : 2 "blocks"
   User_Information         : none

            Display Keyed_File_Properties              1985-10-03
NOS/VE Keyed File Utilities 1.1 85259                    10:31:23

File_attributes and structural_properties of
 $MAIN_FILE

   Block_Count                 : 1
   Compression_Procedure_Name  :
   Creation_Date               : 3/25/85 15:50:14.274
   Data_Padding                : 0 "%"
   Embedded_Key                : yes
   File_Organization           : indexed_sequential
   Index_Levels "current"      : 0
   Index_Level_Overflow        : no
   Index_Padding               : 0 "%"
   Key_Length                  : 5 "bytes"
   Key_Position                : 0
   Key_Type                    : uncollated
   Maximum_Record_Length       : 80 "bytes"
   Minimum_Record_Length       : 5 "bytes"
   Record_Type                 : undefined
   Ruined_Flag                 : off
```

DISPLAY_KEYED_FILE Command

Purpose Formats and displays the contents of a keyed file.

For Better Performance

Do not use DISPLAY_KEYED_FILE while the file is being updated. DISPLAY_KEYED_FILE must wait until all (shorten, append, modify) updates in progress have completed before it begins, and then it forces all file updates to wait until it completes.

Format

DISPLAY_KEYED_FILE or
DISKF
INPUT=file
OUTPUT=file
FORMAT=keyword_value
DISPLAY_OPTIONS=list of keyword_value
BLOCK_LIST=list of range of integer
STATUS=status_variable

Parameters

INPUT or I

File whose contents are to be displayed. You must have at least read permission to the file. This parameter is required.

OUTPUT or O

File to which the formatted display is written. If you omit the OUTPUT parameter, the display is written to file $OUTPUT.

FORMAT or F

List of one or more keyword values indicating the representation used for the contents of records.

ASCII or A	ASCII characters.
HEXADECIMAL or H	Hexadecimal digits.
ALL	Both ASCII characters and hexadecimal digits. (No other formats can be specified with ALL.)

If you omit the FORMAT parameter, the representation used is ASCII.

DISPLAY_OPTIONS or DISPLAY_OPTION or DO

List of one or more keyword values indicating the types of information to be displayed.

MAP or M	Cross-reference of all blocks
TABLES or T	Formatted contents of internal tables
INDEX_BLOCKS or IB or I	Index records
DATA_BLOCKS or DB or D	Data records
EMPTY_BLOCKS or EB or E	Block numbers of empty blocks
ALL or A	All the preceding options. (No other display options can be specified with ALL.)

The default value depends on whether the BLOCK_LIST parameter is specified. If the BLOCK_LIST parameter is not specified, the default value is MAP. If the BLOCK_LIST parameter is specified, the default value is ALL.

BLOCK_LIST or BL

Optional list of block numbers indicating the blocks to be displayed. The blocks are displayed in the order specified in the list.

You can specify from 1 through 999 block numbers and ranges of block numbers. Block numbers range from 0 through 4398046511103 ([2**42] - 1).

The BLOCK_LIST parameter does not limit the blocks in the MAP cross-reference.

If you omit the BLOCK_LIST parameter, the command applies to all blocks in the file.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- A dump of even a small keyed file produces a very long listing. So it is recommended that you first get a cross-reference listing of the file (DISPLAY_OPTION=MAP) so that you can limit the file dump to only the pertinent information.

  The parameters that limit the file dump are FORMAT, DISPLAY_OPTIONS, and BLOCK_LIST.

- Do not specify FORMAT=ALL unless you require both ASCII and hexadecimal representation; ALL doubles the number of lines required to list record contents.

- The DISPLAY_OPTIONS parameter specifies the types of information dumped.

  The MAP keyword produces a cross-reference. The cross-reference lists the header, index, and data block numbers for each nested file and the header and index block numbers for each alternate key. It also lists the empty blocks.

  The TABLES keyword produces a listing of these internal tables:

  Block header
  File configuration
  Rasp configuration
  Segment control
  Rasp list
  Alternate-key definition

  If the EMPTY_BLOCKS keyword is specified, each empty block encountered adds a line to the display, containing the block number of the empty block. If the EMPTY_BLOCKS keyword is omitted, empty blocks encountered are ignored.

- In general, the DISPLAY_OPTIONS and BLOCK_LIST parameters work together to limit the information in the display. The display includes only the types of information specified that apply to the blocks specified. The only exception is for the MAP keyword; whenever it is specified, a cross-reference for all blocks is displayed.

- If the file specified on the command is not a keyed file, DISPLAY_KEYED_FILE returns the warning status, AAE$FILE_IS_NOT_A_KEYED_FILE.

- If the file specified on the command does not exist, DISPLAY_KEYED_FILE returns the warning status, AAE$FILE_DOES_NOT_EXIST.

Examples

This command writes a cross-reference of the contents of file $USER.ISFILE on file ISMAP:

```
/display_keyed_file input=$user.isfile output=ismap
```

Assume that using the cross-reference from the previous example, you decide to dump the data records from blocks 6 and 7 and blocks 9 through 15 in ASCII format. To do so, you enter this command:

```
/display_keyed_file input=$user.isfile ..
../output=isdump display_option=data_blocks ..
../block_list=(6,7,9..15)
```

You could then print the listing on file ISDUMP.

Copying To or From a Keyed File

To copy data records to or from a keyed file, use the COPY_KEYED_FILE command. It can:

- Copy data records from a sequential file to a keyed file
- Copy data records from a keyed file to another keyed file with different attributes
- Duplicate an existing keyed file
- Copy data records from a keyed file to a sequential file
- Add data records to an existing keyed file.

The first three operations are shown in later examples in this chapter. The first operation (copying from a sequential file to a keyed file) is shown as part of keyed-file creation and the second and third operations (copying a keyed file to another keyed file) are shown as part of keyed-file re-creation. The last two operations are described here.

Copying Data Records From a Keyed File to a Sequential File

A COPY_KEYED_FILE command can copy records from a keyed file to a sequential file. It reads records sequentially from the input file.

A sequential read from an indexed-sequential file reads records in ascending order by primary-key value. For example, the following command copies the records in an indexed-sequential file to the sequential file, $OUTPUT. The records are listed in order by their primary key (the restaurant name).

```
/copy_keyed_file input=$user.restaurants output=$output
Arnold's            Casual          Pizza
Burger Duke         Casual          Hamburgers
Gung Ho             Casual          Chinese
  .                   .               .
  .                   .               .
  .                   .               .
```

(A file copied to $OUTPUT must contain only displayable characters.)

Adding Data Records to an Existing Keyed File

To add data to an existing keyed file, you must specify $ASIS or $EOI as the file position designator on the output file reference. Otherwise, COPY_KEYED_FILE opens the output file at its beginning-of-information (BOI) and discards all existing data in the file.

For example, the following command adds the data records on file $USER.NEW_MEMBERS to the existing keyed file $USER.MEMBERSHIP:

```
/copy_keyed_file input=$user.new_members ..
../output=$user.membership.$eoi
```

When COPY_KEYED_FILE finds an error (such as a duplicate primary-key value), it terminates with only part of the records added. The following shows the error messages you receive when this happens:

```
/copy_keyed_file, $user.new_members, temporary_file.$eoi
--ERROR-- File TEMPORARY_FILE already contains the key of this
AMP$PUT_NEXT operation -- primary_key = 96070
--FATAL-- File TEMPORARY_FILE: COPY_KEYED_FILE encountered an
error while calling AMP$PUT_RECORD.  Some of the copy has
occurred, and processing stops.
```

All of the records in the input file up to the record with the duplicate key value have been added to the output file. To add the rest of the records, you must create another input file containing only those records.

Because of the possibility of duplicate primary-key values, you may want to add the data to a temporary copy of the output file. If the operation succeeds, you would then replace the permanent copy with the temporary copy. For example:

```
/copy_keyed_file, $user.output_file, temporary_file
/copy_keyed_file, $user.new_records, temporary_file.$eoi
/copy_keyed_file, temporary_file, $user.output_file
```

COPY_KEYED_FILE Command

Purpose Performs a record-by-record copy.

Format

COPY_KEYED_FILE or
COPKF
INPUT=file or (file, nested_file_name)
OUTPUT=file or (file, nested_file_name)
PRESERVE_KEY_DEFINITIONS=boolean
STATUS=status_variable

Parameters

INPUT or I

File to be copied. You must have at least read permission to the file. This parameter is required.

To specify a nested file, enclose the file reference followed by the nested-file name in parentheses. If you omit the nested-file name, each nested file in the keyed file is copied.

COPY_KEYED_FILE positions the file before the copy according to the open position specified for the file. If a file position is not specified on the file reference, the OPEN_POSITION attribute is used. (The default OPEN_POSITION attribute value is $BOI.)

If the open position is $EOI or $ASIS, only the file attributes are copied; no records are copied from the input file.

OUTPUT or O

File to which the input file is copied. You must have at least append permission to the file. The default output file is the standard file $OUTPUT.

If the INPUT parameter specifies a nested-file reference, the OUTPUT parameter can specify a nested-file reference. (This copies one nested file; you cannot copy multiple nested files to a single nested file or to a sequential file.)

To specify a nested file, enclose the file reference followed by the nested-file name in parentheses.

Do not specify the nested-file name $MAIN_FILE on the OUTPUT parameter when the open_position of the output file is $BOI. (This requests deletion of $MAIN_FILE which is not allowed.)

PRESERVE_KEY_DEFINITIONS or PKD

Indicates whether the alternate-key definitions from the input file (if any) are copied to the output file.

TRUE or ON or YES	Apply alternate-key definitions.
FALSE or OFF or NO	Do not apply alternate-key definitions.

If PRESERVE_KEY_DEFINITIONS is omitted, the alternate-key definitions are copied.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

The first error returned by COPY_KEYED_FILE is stored in the specified status variable; any subsequent error messages are written to the $ERRORS file.

Remarks

- The INPUT and OUTPUT parameters cannot specify the same file cycle unless the parameters specify different nested files in the file cycle.
- COPY_KEYED_FILE supports copying to and from files with sequential and keyed-file organizations. It does not support copying to or from byte-addressable files.
- If the INPUT or OUTPUT file could be shared by more than one instance of open, you should attach the file for exclusive access (SHARE_MODE=NONE) before the copy. This prevents other tasks from locking records, which would cause COPY_KEYED_FILE to terminate.
- COPY_KEYED_FILE reads records sequentially using the CYBIL procedure AMP$GET_NEXT. It reads records from the input file until it reads an end-of-partition or end-of-information delimiter.

  As each record is read, COPY_KEYED_FILE writes the record sequentially to the output file using the CYBIL procedure AMP$PUT_NEXT.

- COPY_KEYED_FILE writes statistics to $ERRORS if requested by the respective MESSAGE_CONTROL attributes of the input and output files. It writes the output file statistics before the input file statistics. (For a sequential file, no statistics are written because the MESSAGE_CONTROL attribute has no effect for sequential files.)

New File

- If the output file is a new file (a file that has never been opened), the output file is given the preserved attributes of the input file that have not been defined for the output file.

  Temporary attributes are not copied.

  If no attributes have been defined for the output file (no SET_FILE_ATTRIBUTES commands have been executed for the file), the new output file is given all attributes of the input file with the following exception:

  The RING_ATTRIBUTES attribute of the input file is not given to the output file. The output file is given the RING_ATTRIBUTES attribute of the caller of the COPY_KEYED_FILE command.

Existing File

- When copying to an existing file, the file attributes of the output file are not changed. The copy performed depends on the output file position as follows:

  - If the file position is $BOI, the output file is overwritten. All output file data and alternate keys are discarded.

  - If the file position is $ASIS or $EOI, the files are merged. The records already existing in the output file are not deleted or replaced.

    If the output file is a keyed file, the primary-key value for each input record is entered, if appropriate, in the alternate indexes already existing in the file.

LIST File

- When copying to a file whose FILE_CONTENTS attribute value is LIST, COPY_KEYED_FILE inserts a space character at the beginning of each record to serve as the carriage control character.

  The FILE_CONTENTS attribute value of a keyed file cannot be LIST.

Fixed-Length Records

- When the output file has fixed-length (F) records, COPY_KEYED_FILE pads input records shorter than the output record length. It pads using the character specified by the PADDING_CHARACTER attribute of the output file. (The default is the space character.)

Differing EMBEDDED_KEY Attributes

- When the input file has nonembedded keys, COPY_KEYED_FILE prefixes the key value to the record data when it reads each record. When the output file has nonembedded keys, COPY_KEYED_FILE assumes that the key value is prefixed to the input record data.

  For example, suppose FILE1, a file with 3-byte nonembedded keys, contains this record:

Key Value	Record Data
KEY	DATA

  Assume you copy FILE1 to FILE2, a file with embedded keys (or a sequential file). The record is written to FILE2 as:

Record Data
KEYDATA

Next, if you copy either FILE1 or FILE2 to FILE3, a file with 1-byte nonembedded keys, the record is written to FILE3 as:

Key Value	Record Data
K	EYDATA

- When the EMBEDDED_KEY attribute differs for the input and output files and PRESERVE_KEY_DEFINITIONS=TRUE is specified, COPY_KEYED_FILE adjusts the alternate-key positions before applying the alternate-key definitions to the output file. The following example shows a copy from embedded to nonembedded:

- Suppose FILE1 has a 5-byte embedded primary key and an alternate key that begins at byte 10. FILE2 is defined with a 5-byte nonembedded primary key. This command is executed:

```
copy_keyed_file file1 file2 ..
  preserve_key_definitions=yes
```

  COPY_KEYED_FILE stores the first 5 bytes of each input record as the primary-key value and writes the rest of the input record as the output record. It then changes the alternate-key position from byte 10 to byte 5. (It subtracts the primary-key length from the old alternate-key position.)

  A copy from nonembedded to embedded works the same way except that it adds (instead of subtracts) the primary-key length to the alternate-key position.

  If an alternate key overlaps the primary key, the alternate-key definition is no longer valid in a file with a nonembedded key. COPY_KEYED_FILE does not apply such an alternate-key definition; it sends a message to notify you of the overlap.

- COPY_KEYED_FILE cannot merge files when the EMBEDDED_KEY attributes of the input and output files differ. Thus, when the EMBEDDED_KEY attributes differ, the file position indicator on the output file reference cannot be $ASIS or $EOI.

- If you prefer that the nonembedded key values be discarded rather than prefixed to the data, use FMU to copy the file, instead of COPY_KEYED_FILE. For more information, see chapter 6.

Alternate Keys

- After the data records have been copied to a keyed file, any alternate-key definitions in the input file are applied to the output file (assuming the command does not specify PRESERVE_KEY_DEFINITIONS=FALSE).

- COPY_KEYED_FILE cannot preserve the first-in, first-out ordering of duplicate values in an alternate index.

  When COPY_KEYED_FILE builds an alternate index, it reads all records in the file sequentially. This means that, for a direct-access file, the records are read in random order and so duplicate values are stored in random order. For an indexed-sequential file, the records are read in order by primary-key value. Thus, each duplicate value is found and stored in primary-key order.

  First-in, first-out ordering is still in effect for later updates to the file.

  To copy a keyed file and keep the first-in, first-out ordering of duplicate key values, use a command that performs a byte-by-byte copy, such as COPY_FILE, BACKUP_PERMANENT_FILE, or CHANGE_KEYED_FILE.

- COPY_KEYED_FILE cannot copy a user-defined collation table directly from the input file to the output file. It gets the collation table name from the input file and then reloads the collation table from the program library list.

  So, before copying a keyed file with a user-defined collation table, you must add the object library containing the collation table to the program library list using a SET_PROGRAM_ATTRIBUTES command. For more information, see appendix E.

Nested Files

- COPY_KEYED_FILE copies a single nested file when its INPUT parameter specifies a nested-file name. Otherwise, it copies all nested files in the input file.
- COPY_KEYED_FILE cannot copy multiple nested files (all nested files in the input file) to a single nested file or to a sequential file.
- Do not specify the nested-file name $MAIN_FILE on the OUTPUT parameter when the open position of the output file is $BOI. (This requests deletion of $MAIN_FILE which is not allowed.)
- COPY_KEYED_FILE creates a new nested file when a nested-file name is specified on the OUTPUT parameter and the open position of the output file is $BOI. It creates the nested file with the attributes of the input nested file. If the input file is a sequential file, COPY_KEYED_FILE cannot create the nested file so it terminates with a fatal error.
- COPY_KEYED_FILE requires append, shorten, and modify permissions to create or replace a nested file.
- COPY_KEYED_FILE merges the records of the input file with those of the output nested file when the open position of the output file is $ASIS or $EOI.
- To merge the records from one nested file into another nested file in the same file, you must copy the records from the input nested file to a temporary file and then copy the temporary file to the output nested file.

  For example, to merge $MAIN_FILE with NESTED_FILE_1 in file $USER.KEYED_FILE:

```
/copy_keyed_file,($USER.KEYED_FILE, ..
$MAIN_FILE),temp
/copy_keyed_file,temp,($USER.keyed_file.$eoi, ..
nested_file_1)
```

- When copying all nested files in the input file (no nested-file name is specified on the INPUT parameter), the copy performed depends on the open position of the output file, as follows:

  - If the open position is $BOI, the contents of the output file are discarded and all input nested files are copied to the output file.

  - If the open position is $ASIS or $BOI, the contents of the input and output files are merged. If an output nested file exists with the same name as an input nested file, the nested files are merged; otherwise, the input nested file is created in the output file. (All nested files that existed in the output file before the copy remain in the output file after the copy.)

Examples This command copies the keyed file .YOUR.ISFILE to the keyed file $USER.ISFILE. It discards any data or alternate keys on $USER.ISFILE and then copies the data and alternate keys from .YOUR.ISFILE to $USER.ISFILE.

```
copy_keyed_file .your.isfile $user.isfile
```

This command copies the keyed file $USER.ISFILE to the next cycle of the file. It does not copy the alternate-key definitions.

```
copkf $user.isfile $user.isfile.$next pkd=no
```

This command copies one nested file to a second nested file. (If the second nested file does not exist, it is created, identical to the first nested file.)

```
copy_keyed_file, ..
 input=($user.direct_access_file, nested_file_1) ..
 output=($user.direct_access_file, nested_file_2)
```

These commands create a new file cycle (cycle 2) containing three nested files. The first command creates the default nested file $MAIN_FILE containing the records from SEQUENTIAL_FILE. The second and third commands create the nested files NF1 and NF2, respectively, each identical to the corresponding nested file in cycle 1.

```
copy_keyed_file,..
 sequential_file $user.keyed_file.2
copy_keyed_file,..
 ($user.keyed_file.1,nf1) ($user.keyed_file.2,nf1)
copy_keyed_file,..
 ($user.keyed_file.1,nf2) ($user.keyed_file.2,nf2)
```

Creating a Keyed File

You can create a keyed file using SCL commands or by using the CREATE_KEYED_FILE utility described in chapter 8.

Creating a keyed file using SCL commands involves three steps:

1. Associate a file cycle with a set of file attributes.
2. Copy data to the new keyed file.
3. Create optional alternate keys.

It is most efficient to create the alternate keys after the data has been copied to the file. However, step 3 can precede step 2. (You can create alternate keys before copying data to the file).

The first step, associating the file name with a set of file attributes, is described in the next subsection.

After defining the attributes for the new keyed file, copying data to a new keyed file (step 2) requires these steps:

1. Enter the data records in a sequential file if the data has not already been captured. (If the data records require reformatting, use FMU as described in chapter 10 of this manual.)
2. Create the keyed file by copying the data records to it. This can be done by executing a COPY_KEYED_FILE command.

   (When using COPY_KEYED_FILE to copy to an empty file that already contains alternate-key definitions, specify $EOI or $ASIS as the output file open position. Otherwise, the alternate-key definitions are evicted from the output file.)

   Or, for an indexed-sequential file, a SORT or MERGE command can sort and copy the data records to the file. This is useful if:

   a. The input records are not sorted by the primary key.

   b. The data records are in more than one file.

   (For more information, see the Sort/Merge TO parameter description in chapter 2.)

The third step, defining alternate keys, is described in chapter 3.

Defining Keyed-File Attributes

The NOS/VE command SET_FILE_ATTRIBUTES defines the attributes of a file. The first parameter of the command specifies the file cycle. Each subsequent parameter specifies the value of a file attribute.

The SET_FILE_ATTRIBUTES command can set any of the file attributes. This section describes only the keyed-file attributes. For the complete SET_FILE_ATTRIBUTES command format, see the NOS/VE Commands and Functions manual.

NOTE

Most attributes have a default value that is used if you do not specify the attribute on the SET_FILE_ATTRIBUTES command. However, the default value is sometimes inappropriate for keyed files. Therefore, it is recommended that you explicitly specify a value for all relevant keyed-file attributes.

To create a keyed file, you specify a keyed-file organization as the file_organization attribute.

For an indexed-sequential file, specify:

```
FILE_ORGANIZATION=INDEXED_SEQUENTIAL
or, abbreviated, FO=IS
```

For a direct-access file, specify:

```
FILE_ORGANIZATION=DIRECT_ACCESS
or, abbreviated, FO=DA
```

The FILE_ORGANIZATION attribute is a preserved attribute.

The other keyed-file attributes define record attributes, primary-key attributes, file structure attributes, and processing attributes.

Record Attributes

These attributes describe the data records to be copied to the keyed file. To determine the attribute values of the sequential file containing the data to be copied to the keyed file, use a DISPLAY_FILE_ATTRIBUTES command, such as:

```
/display_file_attributes data_file ..
../display_options=(record_type, maximum_record_length, ..
../minimum_record_length)
```

NOTE

The record attributes are all preserved attributes, that is, the attribute value is stored with the file when the file is first opened and cannot be changed thereafter.

Each parameter description begins with the parameter name followed by its abbreviation in parentheses.

RECORD_TYPE or RT

Record type: FIXED (F), VARIABLE (V), or UNDEFINED (U). The default is UNDEFINED. (For keyed files, the record types VARIABLE and UNDEFINED are processed as the same and the record type TRAILING_CHARACTER_DELIMITED [T] is not supported.)

MAXIMUM_RECORD_LENGTH or MAXRL

Maximum number of bytes in a data record (from 1 through 65497). This parameter is required.

MINIMUM_RECORD_LENGTH or MINRL

Minimum number of bytes in a data record (from 0 through 65497).

If the RECORD_TYPE is FIXED, the default minimum record length is 0. However, the length of all fixed-length records must be the MAXIMUM_RECORD_LENGTH value.

When the records are variable-length and the key is embedded, the default is the sum of the KEY_POSITION and KEY_LENGTH values. The default for variable-length records with a nonembedded key is 1.

For variable-length records, you should explicitly specify this attribute. The minimum record length must include:

- The primary-key field
- All fixed-length alternate-key fields (or their sparse-key control characters) unless the key repeats to the end of the record.

Primary-Key Attributes

These attributes define the primary key of the new file. See chapter 5 for more information on primary keys.

NOTE

The primary-key attributes are all preserved attributes, that is, the attribute value is stored with the file when the file is first opened and cannot be changed thereafter.

Each parameter description begins with the parameter name followed by its abbreviation.

EMBEDDED_KEY or EK

SCL boolean value indicating whether the primary key is part of the record data (embedded) or separate from the record data (nonembedded). The default is TRUE (embedded keys).

KEY_LENGTH or KL

Integer specifying the primary-key length in bytes (for integer keys, from 1 through 8; for other key types, from 1 through 255). This parameter is required.

KEY_POSITION or KP

Position of the leftmost byte in the primary key (specified only if the key is embedded). The byte positions in a record are numbered from the left, beginning with 0. The default is 0.

KEY_TYPE or KT

Primary key type: UNCOLLATED (UC), INTEGER (I), or COLLATED (C). The default is UNCOLLATED.

For a direct-access file, any value specified for the KEY_TYPE attribute is ignored; the KEY_TYPE attribute value for direct-access files is always UNCOLLATED.

COLLATE_TABLE_NAME or CTN

Name of the collating sequence by which collated keys are ordered (required if the KEY_TYPE is COLLATED).

The name can be the name of a NOS/VE predefined collating sequence or a user-defined collating sequence (an entry point in an object library). See appendix E for more information.

If a collation table name has been specified, the collation table is loaded when the file is first opened; however, it is not used unless the file is an indexed-sequential file with a collated primary key.

File Structure Attributes

The file structure attributes define characteristics of the internal file structure. Some attributes are common to all keyed-file organizations while others apply to only one organization. See chapter 5 for a description of keyed-file structure.

NOTE

The FILE_LIMIT attribute sets a limit on the maximum file length in bytes. If the length of a keyed file reaches its FILE_LIMIT value, the ruined flag is set. (This prevents access to the file data so the file must be re-created using COPY_KEYED_FILE.)

The default FILE_LIMIT value is its maximum value (2^{42}-1) so, for keyed files, use the default FILE_LIMIT value.

Each parameter description begins with the parameter name followed by its abbreviation.

RECORD_LIMIT or RL

Maximum number of data records allowed in each nested file in the file (integer greater than 0).

The default value is the maximum allowed value (2^{42}-1). Thus, you should specify this attribute only when you want to limit the number of records in the file to less than the maximum.

You can increase the RECORD_LIMIT value with a CHANGE_FILE_ATTRIBUTE command even after the file has been opened. For more information, see the NOS/VE System Usage manual.

MAXIMUM_BLOCK_LENGTH or MAXBL

Number of bytes in each block (integer from 1 through 16777215). If the value is less than the maximum record length, it is increased to that value. Then, if the value is not a power of 2 between 2048 and 65536, it is changed as follows:

- A value less than 2048 is increased to 2048 (the minimum allocation unit).
- A value between 2048 and 65536, but not a power of 2, is increased to the next power of 2 (4096, 8192, 16384, 32768, or 65536).
- A value greater than 65536 is decreased to 65536.

The minimum block length is one page if the MAXIMUM_BLOCK_LENGTH attribute is not specified. The minimum block length is 2048 bytes if the MAXIMUM_BLOCK_LENGTH is specified.

NOTE

If the file will be shared by more than one concurrent instance of open and forced-writing will be used (the FORCED_WRITE attribute is either TRUE or FORCED_IF_STRUCTURE_CHANGE), its block size should be a multiple of a system page size. This ensures that more than one instance of open is not updating blocks in the same page; otherwise, a forced-write operation could write a page to mass storage that contains partially-altered blocks. (A warning message is issued if this situation exists.)

It is recommended that you do not specify the block length with the MAXIMUM_BLOCK_LENGTH attribute, but rather, allow the system to calculate the block length using values specified by the following parameters:

NOTE

The following parameters do not set limits; their values are used only as guidelines for determining the block length when the file is created.

AVERAGE_RECORD_LENGTH or ARL

Estimated median record length, in bytes, of the data records to be stored in the file. (The length should not include a nonembedded key. If a compression procedure is used, the average record length should be that of the compressed record.) If you omit this parameter, the system uses the arithmetic mean between the maximum and minimum record lengths in its calculation of the block size.

ESTIMATED_RECORD_COUNT or ERC

Estimated number of data records to be stored in the file. If you omit this parameter, the system uses in its calculation of the block size either the RECORD_LIMIT value or, if that parameter is omitted, the value 100000.

Block Length Guideline Attributes for Indexed-Sequential Files Only

INDEX_LEVELS or INDEX_LEVEL or IL

Target number of index levels for the file (0 through 15). The default value is 2.

RECORDS_PER_BLOCK or RPB

Estimated number of data records to be stored in each data block. If you omit this parameter, the system uses the value 2 in its calculation of the block size.

File Structure Attributes for Indexed-Sequential Files Only

DATA_PADDING or DP

Percentage of data-block space left empty when a block is created (integer from 0 through 99). The default is 0. The percentage must allow for storage of at least one maximum-length record per block.

INDEX_PADDING or IP

Percentage of index-block space left empty when a block is created (integer from 0 through 99). The default is 0. The percentage must allow for storage of at least three index records per block. (The index record length is the key length plus 4.)

File Structure Attributes for Direct-Access Files Only

INITIAL_HOME_BLOCK_COUNT or IHBC

Number of home blocks to be created in the file (1 through 2^{42}-1). This attribute must be specified when creating a direct-access file. For more information, see the Direct-Access File Structure discussion in chapter 5.

HASHING_PROCEDURE_NAME or HPN

Name of the hashing procedure to be executed with this file. The default hashing procedure is the one provided by the system AMP$SYSTEM_HASHING_PROCEDURE. For more information, see the hashing procedure discussion in chapter 5.

Processing Attributes

The following attributes set keyed-file processing options.

COMPRESSION_PROCEDURE_NAME or CPN

Name of the data compression or encryption procedure (preserved attribute).

The attribute has no default value. Unless a procedure is specified when the file is created, no compression procedure is used.

The name must be either the name of the system-defined compression procedure (AMP$RECORD_COMPRESSION) or the name of an entry point in the current program library list.

For more information on data compression and encryption, see the FORTRAN Language Definition or CYBIL Keyed-File and Sort/Merge Interfaces manual.

ERROR_LIMIT or EL

Maximum number of nonfatal (trivial) errors that can occur before the nonfatal errors cause a fatal error. The default value is 0, meaning no limit.

ERROR_LIMIT is a temporary attribute; its value can be changed each time the file is used.

COPY_KEYED_FILE and COPY_FILE do not copy temporary attributes so, if the ERROR_LIMIT is to be greater than 0, you must set the attribute explicitly for the file copy.

LOCK_EXPIRATION_TIME or LET

Number of milliseconds between the time a lock is granted and the time that it could expire (integer from 0 through 604,800,000).

The default value is 60,000 milliseconds (60 seconds). For an unlimited expiration time, set the attribute to 0. This attribute value can be changed by a CHANGE_FILE_ATTRIBUTES command. Locks are described in the COBOL Usage, CYBIL Keyed-File and Sort/Merge Interfaces, and FORTRAN Language Definition manuals.

MESSAGE_CONTROL or MC

List of one or more keyword values indicating the additional information written to the $ERRORS file besides fatal and catastrophic error messages.

TRIVIAL_ERRORS or T	Nonfatal-error messages
MESSAGES or M	Informative messages
STATISTICS or S	Statistical messages
NONE	Suppress nonfatal-error, informative, and statistical messages.

The default value is NONE.

MESSAGE_CONTROL is a temporary attribute; its value can be changed each time the file is used.

COPY_KEYED_FILE and COPY_FILE do not copy temporary attributes so the MESSAGE_CONTROL value will be NONE unless you set the attribute explicitly for the file copy.

Recovery Attributes

The recovery attributes define options that enable recovery of the keyed file. For more information, see chapter 9.

FORCED_WRITE or FW

SCL boolean or keyword value indicating when the system copies modified blocks to mass storage.

TRUE

Write modified blocks immediately.

FALSE

Allow modified blocks to remain in memory until the next flush or close request.

FORCED_IF_STRUCTURE_CHANGE or FISC

Write modified blocks immediately if the change affects more than one block.

The default value is FALSE.

For Better Performance

To prevent serious performance degradation, the FORCED_WRITE attribute should be set to FALSE if the LOGGING_OPTIONS attribute is set to ENABLE_MEDIA_RECOVERY.

LOG_RESIDENCE or LR

Catalog path for the update recovery log for the keyed file. The log must be created by the Administer_Recovery_Log utility described in chapter 9.

Any number of keyed files can use the same log; the log entries for a keyed file are identified by a unique identifier (a signature) for the file to which they apply.

Log entries are not written for the file unless its LOGGING_OPTIONS attribute specifies ENABLE_MEDIA_RECOVERY. If so, the default log is $SYSTEM.AAM.SHARED_RECOVERY_LOG.

NOTE

It is not recommended that the default log, $SYSTEM.AAM.SHARED_RECOVERY_LOG, be used extensively for logging update operations. In general, you should specify a different LOG_RESIDENCE for vital applications. This enables you to isolate the effects of a media failure on the log.

Also, whenever you change the LOG_RESIDENCE of an existing file to a log other than the default log, you should immediately backup the file; otherwise, no entries are logged. If a backup has not been done since the change and the file is damaged, the RECOVER_FILE_MEDIA subcommand of the AMP$RECOVER_KEYED_FILE call cannot execute successfully for the file.

LOGGING_OPTIONS or LOGGING_OPTION or LO

Set of options enabling use of the request keyed-file recovery options. (For more information, see chapter 9.)

ENABLE_PARCELS or EP

For future implementation.

ENABLE_MEDIA_RECOVERY or EMR

Indicates that an update recovery log is to be maintained for the keyed file.

ENABLE_REQUEST_RECOVERY or ERR

Indicates that the automatic close upon task abort removes any partially-completed update operation caused by a system failure (see Protecting Your Keyed Files described in chapter 9).

ALL

All logging options are enabled.

NONE

No logging options are enabled.

The default value is NONE, no logging options enabled.

For Better Performance

Whenever you change the LOGGING_OPTIONS attribute of an existing file, you should immediately backup the file; otherwise, no entries are logged. If a backup has not been done since the change and the file is damaged, the RECOVER_FILE_MEDIA subcommand of the AMP$RECOVER_KEYED_FILE call cannot execute successfully for the file.

Also, to prevent serious performance degradation, the FORCED_WRITE attribute should be set to FALSE if the LOGGING_OPTIONS attribute is set to ENABLE_MEDIA_RECOVERY.

Keyed-File Creation Example

Let's assume that you have been asked to convert the membership records file from a sequential file to an indexed-sequential file. The format of the data in each record is to stay the same; only the file organization is to change. Data reformatting would require use of FMU as described in chapter 10.)

For the purposes of this example, assume that a sequential character data file exists with these specifications:

File reference:	$USER.MEMBER_RECORDS
Record length:	125 to 150 bytes
Primary key:	First six characters of each record
Number of records:	Approximately 5000
Future space estimates:	Replaced records may increase size up to 20%. Number of records may increase up to 25%. During addition of the first 5000 records, the primary key of each new record will always be greater than those of existing records.

Assume also, that you decide that the default attribute values are appropriate for the record type (undefined), embedded key (TRUE), key position (0), and key type (uncollated).

To create the indexed-sequential file named $USER.MEMBERSHIP, you execute these commands:

```
/set_file_attributes ..                    "Attributes of the new
../file=$user.membership ..                "indexed-sequential
../file_organization=indexed_sequential .. "file.
../maximum_record_length=150 ..            "Maximum and minimum
../minimum_record_length=125 ..            "record lengths of the
../key_length=6 data_padding=20 ..         "sequential file.
../index_padding=25 ..
../estimated_record_count=6250             "5000 existing records +
/sort from=$user.member_records ..         "25% growth.
../to=$user.membership ..                  "Sorts the input records
../key=((1..6,ascii,a))                    "by the primary key and
                                           "then writes the records
                                           "to the defined indexed-
                                           "sequential file.
```

If desired, you could next define alternate keys for $USER.MEMBERSHIP using the CREATE_ALTERNATE_INDEXES utility described in chapter 8.

Re-Creating a Keyed File

Keyed file re-creation is required when the ruined flag has been set for a file. It is also recommended when file updates (record additions, deletions, and replacements) have produced an inefficient file structure.

A DISPLAY_KEYED_FILE_PROPERTIES display shows the ruined flag value and structural property values that could indicate an inefficient file structure. The display could show:

Excessive index levels or overflow blocks

A large empty block count

An excessive number of data blocks in relation to the number of data records in the file

If the display shows that the index-level-overflow flag has been set, you must re-create the keyed file to allow record additions.

Keyed-file re-creation is also required when record reformatting is required. It can be done using FMU. For example, suppose the identification number used as the primary key of a file is changed from a 6-character to a 7-character field. The existing records must be reformatted so the identification number is a 7-character field and the file is recreated with a 7-character primary key.

To re-create a keyed file using FMU, see chapter 10 of this manual. This section only describes re-creation using COPY_KEYED_FILE.

To re-create a keyed file for improved file efficiency, you perform two steps:

1. Set the file attributes that are to change for the re-created file.
2. Copy the data from the old keyed file to the new keyed file cycle.

Keyed-File Re-Creation Example

For example, suppose the file structure for the membership file created in the keyed-file creation example has become inefficient. The file specifications are as follows:

File reference:	$USER.MEMBERSHIP.
Record length:	125 to 150 bytes.
Primary key:	First six characters of each record.
Number of records:	Approximately 10000.
Future space estimates:	Replaced records may increase size up to 20%. Number of records may increase up to 100%. The primary key of each new record is not always greater than those of existing records.

Assume that you decide to keep the default attribute values used for the old keyed file: record type (undefined), embedded key (TRUE), key position (0), and key type (uncollated).

To re-create the indexed-sequential file $USER.MEMBERSHIP, you execute these commands:

```
/display_catalog_entry file=$user.membership .. "Display the existing
../display_option=cycles                        "file cycles (only one
membership   1,441,792 bytes                    "cycle [1], exists).

/set_file_attributes ..                         "Set the file attributes
../file=$user.membership.2 ..                   "for cycle 2 that are to
../data_padding=50 index_padding=50 ..          "differ from cycle 1.
../estimated_record_count=20000

/copy_keyed_file input=$user.membership.1 ..    "Copy the file data
../output=$user.membership.2                    "from the old cycle
                                                "to the new cycle.
/display_catalog_entry, $user.membership, ..
../display_option=cycles                        "Display the file cycles.
membership   2,883,584 bytes
-- cycle 1   1,441,792 bytes
-- cycle 2   1,441,792 bytes
/delete_file file=$user.membership.1            "Delete the old cycle.
```

Create_Alternate_Indexes Utility 7

Create_Alternate_Indexes Utility 7

This chapter describes the use of the CREATE_ALTERNATE_INDEXES command utility (also known as the CHANGE_ALTERNATE_INDEXES utility). The utility can create, delete, and display alternate keys in a keyed file.

Alternate key concepts are described in chapter 5 of this manual.

A CREATE_ALTERNATE_INDEXES utility session processes the alternate-key definitions for a single nested file, the nested file specified on the CREATE_ALTERNATE_INDEXES command that begins the session. (If no nested-file name is specified, CREATE_ALTERNATE_INDEXES processes the alternate keys for the default nested file, $MAIN_FILE.)

The CREATE_ALTERNATE_INDEXES command utility is an SCL command utility. As with all SCL command utilities, its use requires three steps:

1. Enter the command (in this case, CREATE_ALTERNATE_INDEXES) to begin the utility session.
2. Enter utility subcommands to direct the utility in the tasks it is to perform.
3. Enter the utility subcommand QUIT to end the utility session.

Any SCL command can be entered in response to a utility prompt.

Creating Alternate Keys

Alternate-key creation requires two steps:

1. Definition of one or more alternate keys.

2. Application of the alternate-key definitions to the file.

Within a CREATE_ALTERNATE_INDEXES session, these steps correspond to the following commands:

1. One or more CREATE_KEY_DEFINITION subcommands to define the attributes of the new alternate keys.

2. Either an APPLY_KEY_DEFINITIONS subcommand to apply the alternate-key definitions or a QUIT APPLY_KEY_DEFINITIONS=YES subcommand to both apply the alternate-key definitions and end the utility session.

Step 2 (application of the key definition) does not inevitably follow step 1 (definition of an alternate key). Step 1 specifies a pending alternate-key definition, that is, an alternate-key definition that has not yet been applied to the file. You can cancel any pending alternate-key definition with a CANCEL_KEY_DEFINITION subcommand.

Any number of subcommands can be entered between the specification of an alternate-key definition and its application to the file. The APPLY_KEY_DEFINITIONS subcommand or parameter applies all definition and deletion requests pending at that time.

Deleting Alternate Keys

Alternate-key deletion is similar to alternate-key creation. It requires two steps:

1. One or more requests to delete alternate keys.
2. Application of the alternate-key deletion requests to the file.

Within a CREATE_ALTERNATE_INDEXES session, these steps correspond to the following commands:

1. One or more DELETE_KEY_DEFINITION subcommands to request the alternate-key deletions.
2. Either an APPLY_KEY_DEFINITIONS subcommand to apply the alternate-key deletion requests or a QUIT APPLY_KEY_DEFINITIONS=YES subcommand to both apply the alternate-key deletion requests and end the utility session.

As with alternate-key creation, step 2 (application of the key deletion request) does not inevitably follow step 1 (requesting deletion of an alternate key). Step 1 defines a pending deletion request, that is, an alternate-key deletion request that has not yet been applied to the file. You can cancel any pending deletion requests with a CANCEL_KEY_DEFINITIONS subcommand.

Displaying Alternate Keys

These methods are available to display existing alternate-key information:

- DISPLAY_KEYED_FILE PROPERTIES: displays all alternate-key definitions that have been applied to a keyed file (described in chapter 6).
- In a CREATE_KEYED_FILE or CHANGE_KEYED_FILE utility session (described in chapter 7):
  - DISPLAY_NESTED_FILE can list the alternate-key names.
  - DISPLAY_RECORDS can display the alternate-key values.

However, to display information about both existing and pending alternate-key definitions, you must use the DISPLAY_KEY_DEFINITIONS subcommand in a CREATE_ALTERNATE_INDEXES session.

The DISPLAY_KEY_DEFINITIONS subcommand has several options. Depending on the parameter values specified, you can:

- Choose the definitions and deletions displayed:
  - By name
  - By state (pending or pending and applied)
- Choose the display content:
  - Brief or full attribute listing
  - Sample records with alternate-key fields marked

Alternate-Key Creation and Deletion Example

The following interactive session illustrates use of the CREATE_ALTERNATE_INDEXES utility:

```
/create_alternate_indexes ..                    "Starts the utility session.
../input=$user.restaurants
creai/display_file_attributes ..                "The alternate keys defined must
creai../file=$user.restaurants ..               "be within the minimum record
creai../display_option=minimum_record_length    "length.
Minimum_Record_Length       : 36
creai/display_key_definitions ..                "Displays the existing alternate
creai../key_names=all                           "key definitions.
              Display_Key_Definitions                 1985-10-03
NOS/VE Keyed File Utilities 1.1 85259                   13:54:09
File = .NVE.USER99.RESTAURANTS

 KEY NAME                              POSITION LENGTH TYPE       STATE
 ------------------------------------ -------- ------ ---------- ----------------
 FOOD                                        15     15 uncollated Exists in file
   Duplicate_Key_Values             : not_allowed
   Null_Suppression                 : no
================================================================================
RECORD 1 ............(in ascii) : B u r g e r   D u k e         H a m b u r g e
                    ( in hex ) : 427572267672204475687620202020486160275726765
FOOD                           :                                  U_U_U_U_U_U_U_
                    (in ascii) : r s           C a s u a l
                    ( in hex ) : 727320202020203617375616C
                               > U_U_U_U_U_U_U_
creai/delete_key_definition ..                  "Requests deletion of the
creai../key_name=food                           "existing alternate key.
creai/create_key_definition key_name=food ..    "Redefines the alternate key.
creai../key_position=15 key_length=15 ..
creai../duplicate_key_value=..
creai../ordered_by_primary_key
creai/create_key_definition ..                  "Defines a new alternate key.
creai../key_name=style key_position=30 ..
creai../key_length=6 duplicate_key_values=..
creai../ordered_by_primary_key
```

```
creai/display_key_definitions                "Displays the pending requests.
              Display_Key_Definitions                  1986-11-03
NOS/VE Keyed File Utilities 1.3 85259                    13:54:59
File = .NVE.USER99.RESTAURANTS

 KEY NAME                                 POSITION LENGTH TYPE       STATE
 ---------------------------------------- -------- ------ ---------- ----------------
 STYLE                                          30      6 uncollated Creation pending
   Duplicate_Key_Values                   : ordered_by_primary_key
   Null_Suppression                       : no
 FOOD                                           15     15 uncollated Creation pending
   Duplicate_Key_Values                   : ordered_by_primary_key
   Null_Suppression                       : no
================================================================================
RECORD 1 ..........(in ascii) : B u r g e r   D u k e         H a m b u r g e
                   ( in hex ) : 427572267672204475687620202020486160627572676 5
STYLE                         :
FOOD                          :                                    U_U_U_U_U_U_U_
                   (in ascii) : r s           C a s u a l
                   ( in hex ) : 72732020202020436173756 16C
                              >               U_U_U_U_U_U_
                              > U_U_U_U_U_U_U_

creai/cancel_key_definition key_name=style    "Cancels a pending definition.
creai/apply_key_definitions                   "Applies the key definitions.
-- File :NVE.USER99.RESTAURANTS : begin deleting alternate key definitions.
-- File :NVE.USER99.RESTAURANTS : alternate key FOOD has been deleted.
-- File :NVE.USER99.RESTAURANTS : end deleting alternate key definitions.
-- File :NVE.USER99.RESTAURANTS : begin creating labels for alternate key
definitions.
-- File :NVE.USER99.RESTAURANTS : finished creating labels for alternate key
definitions.
-- File :NVE.USER99.RESTAURANTS : begin collecting the alternate key values from
the file.
-- File :NVE.USER99.RESTAURANTS : AMP$APPLY_KEY_DEFINITIONS has reached a file
boundary: EOI .
-- File :NVE.USER99.RESTAURANTS : collection of the alternate key values
is complete.
-- File :NVE.USER99.RESTAURANTS : begin sorting the alternate key values.
-- File :NVE.USER99.RESTAURANTS : sorting of the alternate key values completed.
-- File :NVE.USER99.RESTAURANTS : begin building alternate key indexes into the
file.
-- File :NVE.USER99.RESTAURANTS : the FOOD index is being built.
-- File :NVE.USER99.RESTAURANTS : AMP$APPLY_KEY_DEFINITIONS completed building
the alternate indexes into the file.
creai/quit
```

CREATE_ALTERNATE_INDEXES Command

Purpose Begins a CREATE_ALTERNATE_INDEXES utility session to create, delete, and/or display alternate-key definitions in a keyed file.

Format

CREATE_ALTERNATE_INDEXES or
CHANGE_ALTERNATE_INDEXES or
CREATE_ALTERNATE_INDICES or
CHANGE_ALTERNATE_INDICES or
CREATE_ALTERNATE_INDEX or
CHANGE_ALTERNATE_INDEX or
CREAI or
CHAAI
INPUT=file or (file, nested_file_name)
STATUS=status_variable

Parameters

INPUT or I

Keyed file to be processed by the utility. The file permissions required depend on the subcommands entered during the utility as described in the Remarks. This parameter is required.

The operations performed during the session apply to only one nested file. If no nested file is specified on the command, the default nested file, $MAIN_FILE, is used.

To process the alternate-key definitions for a nested file other than $MAIN_FILE, enclose the file reference followed by the nested-file name in parentheses.

If the specified input file does not exist, the command attempts to create it as described in the Remarks.

STATUS

Optional SCL status variable. Use of this variable depends on whether the utility is executed in an SCL block. If it is, the status of each subcommand is stored in the status variable specified on the CREATE_ALTERNATE_INDEXES command and a subcommand error terminates the utility session.

Otherwise, only the status of the CREATE_ALTERNATE_INDEXES command and the QUIT subcommand that ends the session are stored in the status variable; a subcommand error does not terminate the utility session.

Remarks

- The command utility prompt is:

  ```
  creai/
  ```

- In response to the creai/ prompt, you can enter NOS/VE commands and any of these subcommands:

  QUIT
  HELP
  DISPLAY_KEY_DEFINITIONS
  CREATE_KEY_DEFINITION
  DELETE_KEY_DEFINITION
  CANCEL_KEY_DEFINITIONS
  APPLY_KEY_DEFINITIONS

- The CREATE_ALTERNATE_INDEXES utility creates the specified keyed file if:

  - The file does not exist and,
  - A SET_FILE_ATTRIBUTES command has specified the KEY_LENGTH and MAXIMUM_RECORD_LENGTH attributes for the file.

  If the SET_FILE_ATTRIBUTES command defining the new file omits an attribute, the default attribute value is used. However, if it omits the FILE_ORGANIZATION attribute, indexed-sequential organization is used.

- The CREATE_ALTERNATE_INDEXES command does not check your file permissions; the subcommands you enter in the utility session check that you have the required permissions to do the operation.

  To display key definitions, you must have at least read permission; to create, delete, cancel, or apply key definitions, you must have at least the three permissions: append, modify, and shorten.

Examples

This command begins a utility session that displays the alternate-key definitions of keyed file $USER.IS_FILE.

```
/create_alternate_indexes input=$user.is_file
creai/display_key_definitions key_names=all ..
creai../display_options=brief
                Display_Key_Definitions                 1985-10-03
NOS/VE Keyed File Utilities 1.1 85259                     13:58:09
File = :NVE.USER99.IS_FILE

 KEY NAME                              POSITION LENGTH TYPE       STATE
 ------------------------------------ -------- ------ ---------- --------
 ALTERNATE_KEY_1                             0     10 uncollated Exists in
 file
creai/quit      "The APPLY_KEY_DEFINITIONS parameter is not required here
                "because no creation or deletion requests are pending.
```

APPLY_KEY_DEFINITIONS Subcommand

Purpose

Applies the pending alternate-key definition and deletion requests within a CREATE_ALTERNATE_INDEXES utility session.

For Better Performance

Use a batch job to apply key definitions to a large file, not an interactive session. The building of an alternate index can be time-consuming, preventing use of the terminal.

Format

APPLY_KEY_DEFINITIONS or
APPLY_KEY_DEFINITION or
APPKD
ERROR_LIMIT=integer
STATUS=status_variable

Parameters

ERROR_LIMIT or EL

Number of nonfatal (trivial) errors allowed for the apply operation (integer from 0 through 4398046511103 [2^{42} - 1]).

A 0 value indicates no limit; 0 is the default value.

See the Remarks for a description of apply error processing.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- This CREATE_ALTERNATE_INDEXES subcommand applies all pending alternate-key creation and deletion requests to the file. It applies the deletion requests first and then the creation requests.
- The ERROR_LIMIT file attribute value has no effect on the utility. This is so that nonfatal errors (such as typing errors during interactive use) do not terminate the utility session.

  However, you can specify an error limit for the apply operation with the ERROR_LIMIT parameter.

Nonfatal Errors

The two nonfatal (trivial) errors that an apply operation can detect result from improper record data, as follows:

- Duplicate_Key_Value: the duplicate-key-values attribute of the alternate index being built is NOT_ALLOWED, but the apply operation finds an alternate-key value matching an alternate-key value already in the alternate index.
- Sparse_Key_Beyond_EOR: the apply operation is building an alternate index that uses sparse-key control and it finds a record for which an alternate-key value should be included in the index except that the record is too short to provide a complete alternate-key value.

Nonfatal Error Processing

APPLY_KEY_DEFINITIONS keeps a count of the number of times it detects a nonfatal (trivial) error. Each time it increments the count, it checks whether the count has reached the value specified by the ERROR_LIMIT parameter.

- If the error limit has not yet been reached, APPLY_KEY_DEFINITIONS performs the correction processing for the condition as described later.
- If the error limit is reached, APPLY_KEY_DEFINITIONS terminates with a fatal error. The fatal status returned depends on the last nonfatal error dectected:
  - For a Duplicate_Key_Value error, it returns AAE$DUPLICATE_KEY_LIMIT.
  - For a Sparse_Key_Beyond_EOR error, it returns AAE$ERROR_LIMIT_EXCEEDED.

  Before terminating, APPLY_KEY_DEFINITIONS discards all alternate indexes it has built. (Deleted alternate indexes are not restored.)

If APPLY_KEY_DEFINITIONS finds one or more nonfatal errors, but completes its processing before reaching the error limit, it returns the warning status AAE$ERRORS_IN_APPLY.

Correction Processing

As correction processing for a Sparse_Key_Beyond_EOR error, APPLY_KEY_DEFINITIONS does not enter an alternate-key value for the record in the alternate index it is building, even though the sparse-key character indicates that a value should be entered for the record.

As correction processing for a Duplicate_Key_Value error, APPLY_KEY_DEFINITIONS changes the duplicate_key_values attribute of the alternate-key definition from NOT_ALLOWED to ORDERED_BY_PRIMARY_KEY. It then discards the partially-built index and begins building the index again, ordering duplicate alternate-key values by their primary-key value.

Terminate Break

If you enter the terminate_break_character (usually %2 or control-t) during application of alternate-key definitions, you are sent a prompt requesting confirmation of your intentions.

You should then enter a carriage return or any entry other than RUIN FILE (uppercase or lowercase) to continue the application of alternate-key definitions. If the apply operation is allowed to complete, the CREATE_ALTERNATE_INDEXES utility can remove any unwanted alternate-key definitions without harm to the file.

A request to ruin the file is not recommended. No file operation can be performed on a ruined file, and so no data can be retrieved from the file.

Pause Break

Entry of the pause_break_character (usually %1 or control-p) is ignored during application of alternate-key definitions.

Examples

This CREATE_ALTERNATE_INDEXES session attempts to create and apply an alternate key. However, the attempt fails when it finds a duplicate alternate-key value because duplicate key values are not allowed and the specified error limit is 1.

```
/create_alternate_indexes input=$user.is_file
creai/create_key_definition key_name=alternate_key_6 ..
creai../key_position=5 key_length=10
creai/apply_key_definition error_limit=1
-- File :NVE.USER99.IS_FILE : begin creating labels for alternate
key definitions.
-- File :NVE.USER99.IS_FILE : finished creating labels for alternate
key definitions.
-- File :NVE.USER99.IS_FILE : begin collecting the alternate key
values from the file.
-- File :NVE.USER99.IS_FILE : AMP$APPLY_KEY_DEFINITIONS has reached
a file boundary: EOI.
-- File :NVE.USER99.IS_FILE : collection of the alternate key values
is complete.
-- File :NVE.USER99.IS_FILE : begin sorting the alternate key
values.
-- File :NVE.USER99.IS_FILE : sorting of the alternate key values
completed.
-- File :NVE.USER99.IS_FILE : begin building alternate key indexes
into the file.
-- File :NVE.USER99.IS_FILE : the ALTERNATE_KEY_6 index is being
built.
-- File :NVE.USER99.IS_FILE : alternate key ALTERNATE_KEY_6 has been
deleted.
--ERROR-- File :NVE.USER99.IS_FILE : AMP$APPLY_KEY_DEFINITIONS encountered
a duplicate key and found that the nonfatal-error limit had been
reached.  It then discarded any new alternate indexes it had built
(although it cannot restore any alternate indexes it deleted).  Had
ERROR_LIMIT not been reached, the alternate-key definition would have
been modified to allow duplicates.  The duplicate key values relate
to alternate key name = ALTERNATE_KEY_6, primary key = 96070,
alternate_key_value = John Smith.
-- FATAL-- File  :NVE.USER99.IS_FILE : AMP$APPLY_KEY_DEFINITIONS : the
user-declared maximum number of trivial errors has been recorded
since the last OPEN.
creai/quit
```

CANCEL_KEY_DEFINITIONS Subcommand

Purpose Removes a pending request to create or delete an alternate key within a CREATE_ALTERNATE_INDEXES session.

Format

CANCEL_KEY_DEFINITIONS or
CANCEL_KEY_DEFINITION or
CANKD
KEY_NAMES=list of names or keyword_value
STATUS=status_variable

Parameters

KEY_NAMES or KEY_NAME or NAMES or NAME or KN or N

Pending requests to be canceled.

list of names	Cancel the requests for the listed alternate-key names.
ALL	Cancel all requests.

This parameter is required.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- The CANCEL_KEY_DEFINITIONS subcommand can cancel pending creation and deletion requests. A request can be canceled only while it is pending.
- After a creation or deletion request is applied, the CANCEL_KEY_DEFINITIONS subcommand has no effect. To reverse the action of an APPLY_KEY_DEFINITIONS subcommand, you must issue new requests to delete the created alternate key or recreate the deleted alternate key.

Examples

This CREATE_ALTERNATE_INDEXES session requests creation of an alternate key and deletion of another alternate key, cancels the creation request, and finally applies the deletion request.

```
/create_alternate_indexes input=$user.is_file
creai/create_key_definition key_name=alternate_key_4 ..
creai../key_position=5 key_length=2
creai/delete_key_definition key_name=alternate_key_1 ..
creai/cancel_key_definition alternate_key_4
creai/quit apply
-- File :NVE.USER99.IS_FILE : begin deleting alternate
key definitions.
-- File :NVE.USER99.IS_FILE : alternate key
ALTERNATE_KEY_1 has been deleted.
-- File :NVE.USER99.IS_FILE : end deleting alternate
key definitions.
```

CREATE_KEY_DEFINITION Subcommand

Purpose Creates a pending alternate-key definition within a CREATE_ALTERNATE_INDEXES session.

Format

CREATE_KEY_DEFINITION or
CREKD
KEY_NAME=name
KEY_POSITION=integer
KEY_LENGTH=integer
KEY_TYPE=keyword_value
COLLATE_TABLE_NAME=name
DUPLICATE_KEY_VALUES=boolean or keyword_value
NULL_SUPPRESSION=boolean
SPARSE_KEY_CONTROL_POSITION=integer
SPARSE_KEY_CONTROL_CHARACTERS=string
SPARSE_KEY_CONTROL_EFFECT=keyword_value
REPEATING_GROUP_LENGTH=integer
REPEATING_GROUP_COUNT=integer or keyword_value
KEY_GROUP_NAME=name
CONCATENATED_PIECES=boolean
VARIABLE_LENGTH_KEY=string
STATUS=status_variable

Parameters

KEY_NAME or NAME or KN or N

Name of the new alternate key. The name must follow the SCL naming rules. This parameter is required.

KEY_POSITION or POSITION or KP or P

Byte position within the record at which the alternate-key field begins. The byte positions are numbered from the left, beginning with 0. The maximum byte position is 65496. This parameter is required.

KEY_LENGTH or LENGTH or KL or L

Number of bytes (1 through 255) in the alternate-key field. (For variable-length keys, it is the maximum key length.)

The key field (or its sparse-key control character) must be within the minimum record length (except for variable-length keys and fixed-length keys that repeat to the end of record).

This parameter is required.

KEY_TYPE or TYPE or KT or T

Type of the alternate key.

INTEGER or I	Integer key ordered numerically; its leftmost bit is its sign bit. (The INTEGER key type is invalid for variable-length keys).
UNCOLLATED or UC or U	Character key ordered byte-by-byte according to the ASCII collating sequence.
COLLATED or C	Character key ordered byte-by-byte according to the collation table specified by the COLLATE_TABLE_NAME parameter.

If you omit the KEY_TYPE parameter, the key type is UNCOLLATED.

COLLATE_TABLE_NAME or CTN

Name of the collation table used to order the alternate key if its key type is collated.

If the file is an indexed-sequential file with a collated primary key, the collation table for the primary key is used as the default collation table for an alternate key. Otherwise, you must specify a collation table for a collated alternate key.

The collation table can be a NOS/VE predefined collation table or a user-defined collation table. For more information, see appendix E.

DUPLICATE_KEY_VALUES or DKV

Indicates whether duplicate alternate-key values are allowed and, if so, how the duplicate values are ordered.

NOT_ALLOWED or NA or FALSE or OFF or NO	No duplicate values are allowed for the alternate key.
ORDERED_BY_ PRIMARY_ KEY or OBPK or TRUE or ON or YES	Duplicate values are allowed; duplicates are accessed in order by their primary-key value.
FIRST_IN_FIRST_OUT or FIFO	Duplicate values are allowed; duplicates are accessed in the order the values were entered in the index.

If you omit the DUPLICATE_KEY_VALUES parameter, no duplicate values are allowed.

NULL_SUPPRESSION or NS

Indicates whether null alternate-key values should be stored in the alternate index. (The null value is all zeros for integer keys, all blanks for the other key types.)

TRUE or ON or YES	Null values are not included in the index.
FALSE or OFF or NO	All values are included in the index.

If you omit the NULL_SUPPRESSION parameter, all values, including nulls, are stored in the index.

NOTE

The two parameters, SPARSE_KEY_CONTROL_ POSITION and SPARSE_KEY_CONTROL_ CHARACTERS, work together; they must either both be specified or both be omitted. If they are omitted, sparse-key control is not used for the alternate key.

SPARSE_KEY_CONTROL_POSITION or SKCP

Byte position of the sparse-key control character. The position must be within the minimum record length. The byte positions are numbered from the left, beginning with 0. The maximum byte position is 65496.

SPARSE_KEY_CONTROL_CHARACTERS or SKCC

String containing the set of characters with which the sparse-key control character in each record is compared.

SPARSE_KEY_CONTROL_EFFECT or SKCE

Indicates whether a sparse-key control character match causes the alternate-key value to be included in or excluded from the alternate index.

INCLUDE_KEY_VALUE or IKV	The alternate-key value is included in the alternate index.
EXCLUDE_KEY_VALUE or EKV	The alternate-key value is excluded from the alternate index.

If you omit the SPARSE_KEY_CONTROL_EFFECT parameter, INCLUDE_KEY_VALUE is used.

You can specify the SPARSE_KEY_CONTROL_EFFECT parameter only if you specify the SPARSE_KEY_POSITION and SPARSE_KEY_CHARACTERS parameters.

REPEATING_GROUP_LENGTH or RGL

If specified, indicates that each record can contain more than one value for the alternate key.

For a repeating fixed-length key, the value is the distance (1 through 65497 bytes) from the beginning of an alternate-key value to the beginning of the next value for the same alternate key in the same record.

For a repeating variable-length key, specify any integer from 1 through 65497. (The actual value is irrelevant.)

If you omit the REPEATING_GROUP_LENGTH parameter, the alternate key has no more than one value per record.

REPEATING_GROUP_COUNT (RGC)

Indicates where the search for alternate-key values ends.

NOTE

REPEATING_GROUP_COUNT parameter is valid only when you specify the REPEATING_GROUP_LENGTH parameter.

REPEAT_TO_END_OF_RECORD or RTEOR

Search continues to the end of the record.

For a fixed-length key, the repeating group of fields continues to the end of the record. The key value that ends the record is not used if it is shorter than the key length.

For a variable-length key, the record data from the key position to the end of the record is processed as a sequence of key values, separated by delimiter characters. The end of the last value is marked by a delimiter character or by the end of the record.

integer (1 through 65497)

Search continues to the specified limit.

For a fixed-length key, the specified integer is the number of alternate-key values that each record contains. (The value must lie within the minimum record length.)

For a variable-length key, the specified integer is the length, in bytes, of the key field. The contents of the field is processed as a sequence of key values, separated by delimiter characters. The end of the last value is marked by a delimiter character, the end of the field, or the end of the record, whichever occurs first.

If you omit the REPEATING_GROUP_COUNT parameter, the search for values continues until the end of the record.

KEY_GROUP_NAME or KGN

Name of the key group for this key. The key-grouping feature is not currently implemented. The default value for the key-group name is the key name.

CONCATENATED_PIECES or CONCATENATED_ PIECE or CP

Indicates whether the alternate key is a concatenated key.

TRUE (ON or YES)	The key is a concatenated key.
FALSE (OFF or NO)	The key is not a concatenated key.

If you omit the CONCATENATED_PIECES parameter, the key is not a concatenated key.

If you specify CONCATENATED_PIECES=TRUE, the CREATE_KEY_DEFINITION command initiates the CREATE_KEY_DEFINITION subcommand utility. The utility prompt is crekd/ and it processes ADD_PIECE and QUIT subcommands (described in the following pages).

VARIABLE_LENGTH_KEY or VLK

Indicates that the key is a variable_length key by specifying its set of delimiter characters. The set is specified as a string (0 through 256 characters, enclosed in apostrophes).

If the REPEATING_GROUP_LENGTH parameter is omitted, no more than one value for the key is taken from a record. The end of the value is marked by a delimiter character, by the end of the key field (KEY_ LENGTH length), or by the end of the record, whichever occurs first after the KEY_POSITION.

If the REPEATING_GROUP_LENGTH parameter is specified, the record can contain more than one value for the key. Multiple key values are separated by one or more delimiter characters. The REPEATING_ GROUP_COUNT parameter indicates whether the sequence of values continues to the end of the record or is limited to a fixed number of characters.

If VARIABLE_LENGTH_KEY is omitted, the alternate key has fixed-length values.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- The CREATE_KEY_DEFINITION subcommand defines an alternate key but does not apply the definition to the file. The definition remains pending until it is either applied or canceled.

  A definition is applied by either an APPLY_KEY_DEFINITIONS subcommand or an APPLY_KEY_DEFINITIONS=YES parameter on the QUIT subcommand; it is canceled by a CANCEL_KEY_DEFINITIONS subcommand or an APPLY_KEY_DEFINITIONS=NO parameter on the QUIT subcommand.

- The various alternate-key attributes are described in chapter 5.

Incompatible Parameters

These parameters are incompatible:

- REPEATING_GROUP_LENGTH and either of the following:

  DUPLICATE_KEY_VALUES=FIRST_IN_FIRST_OUT
  CONCATENATED_PIECES=TRUE

- VARIABLE_LENGTH_KEY and any of the following:

  KEY_TYPE=INTEGER
  DUPLICATE_KEY_VALUES=FIRST_IN_FIRST_OUT
  CONCATENATED_PIECES=TRUE
  NULL_SUPPRESSION=TRUE
  SPARSE_KEY_CONTROL_POSITION

Collation Table Loading

- If the alternate-key definition defines a collated key, CREATE_KEY_DEFINITIONS searches for the collation-table name as an entry point in the object libraries in the program-library list.

- You must set the program-library list before you enter the utility; you cannot change the object libraries searched from within the utility session. The following command adds an object library to the program-library list:

```
set_program_attributes add_library=file_reference
```

  See appendix E for more information on collation tables.

Examples This CREATE_ALTERNATE_INDEXES session creates and applies an alternate-key definition to file $USER.IS_FILE.

```
/create_alternate_index, input=$user.is_file
creai/create_key_definition, key_name=alternate_key_1 ..
creai../key_position=0, key_length=10
creai/quit, apply
-- File :NVE.USER99.IS_FILE : begin creating labels for alternate key
definitions.
-- File :NVE.USER99.IS_FILE : finished creating labels for alternate
 key definitions.
-- File :NVE.USER99.IS_FILE : begin collecting the alternate key
 values from the file.
-- File :NVE.USER99.IS_FILE : AMP$APPLY_KEY_DEFINITIONS has reached
 a file boundary: EOI
-- File :NVE.USER99.IS_FILE : collection of the alternate key values
 is complete.
-- File :NVE.USER99.IS_FILE : begin sorting the alternate key values.
-- File :NVE.USER99.IS_FILE : sorting of the alternate key values
completed.
-- File :NVE.USER99.IS_FILE : begin building alternate key indexes
into the file.
-- File :NVE.USER99.IS_FILE : the ALTERNATE_KEY_1 index is being built.
-- File :NVE.USER99.IS_FILE : AMP$APPLY_KEY_DEFINITIONS completed
building the alternate indexes into the file.
```

ADD_PIECE Subcommand

Purpose Defines a piece of a concatenated key within a CREATE_KEY_DEFINITION subutility session.

Format

ADD_PIECE or
ADDP
KEY_POSITION=integer
KEY_LENGTH=integer
KEY_TYPE=keyword_value
STATUS=status_variable

Parameters

KEY_POSITION or POSITION or KP or P

Byte position in the record at which the piece begins. The byte positions are numbered from the left, beginning with 0. The maximum byte position is 65496. This parameter is required.

KEY_LENGTH or LENGTH or KL or L

Number of bytes in the piece. The maximum length is 255 bytes. The piece must be within the minimum record length (unless sparse-key control is used). This parameter is required.

KEY_TYPE or TYPE or KT or T

Type of the piece.

INTEGER or I	Integer key ordered numerically.
UNCOLLATED or UC or U	Character key ordered byte-by-byte according to the ASCII collating sequence.
COLLATED or C	Character key ordered byte-by-byte according to the collation table specified by the COLLATE_TABLE_NAME parameter on the CREATE_KEY_DEFINITION command.

The default key type is UNCOLLATED.

STATUS

Optional SCL status variable. If STATUS is specified, the command returns its completion status in the specified variable.

Remarks

- You enter this subcommand in response to this prompt:

```
crekd/
```

  The utility is initiated in response to a CREATE_KEY_DEFINITION subcommand that specifies the CONCATENATED_PIECES=TRUE parameter. To end concatenated-key specification, enter the QUIT subcommand for the CREATE_KEY_DEFINITION utility.

- To define a concatenated key, you must enter an ADD_PIECE subcommand for each piece to be concatenated to the first piece. The first piece is defined by the KEY_LENGTH, KEY_POSITION, and KEY_TYPE parameters on the CREATE_KEY_DEFINITION command.

- A concatenated key can comprise from 2 through 64 pieces. The pieces are concatenated in the order that you enter the ADD_PIECE subcommands that define the pieces.

Examples

This CREATE_ALTERNATE_INDEX session defines an alternate key that concatenates the first, third and fifth bytes of the record in reverse order. It displays the definition and then cancels the request.

```
/create_alternate_index input=$user.is_file
creai/create_key_definition key_name=alternate_key_2 ..
creai../key_position=4 key_length=1 concatenated_pieces=yes
crekd/add_piece key_position=2 key_length=1
crekd/addp kp=0 kl=1
crekd/quit
creai/display_key_definitions
                Display_Key_Definitions                    1985-10-03
NOS/VE Keyed File Utilities 1.1 85259                        14:04:22
File = .NVE.USER99.IS_FILE

 KEY NAME                              POSITION LENGTH TYPE       STATE
 ------------------------------------ -------- ------ ---------- ------------
 ALTERNATE_KEY_2                             4      1 uncollated Creation
 pending
                               piece b       2      1 uncollated
                               piece c       0      1 uncollated
   Duplicate_Key_Values              : not_allowed
   Null_Suppression                  : no
================================================================================
RECORD 1 .......(in ascii) : T h i s   i s   t h e   f i r s t   r e c o r
                 (in hex ) : 546869732069732074686520666972737420726563 6F72
ALTERNATE_KEY_2                  : c_  b_  a_
                      (in ascii) : d .
                      ( in hex ) : 642E
                                   >
creai/quit cancel
```

HELP Subcommand (for the CREATE_KEY_ DEFINITION Utility)

Purpose Provides online help from within a CREATE_KEY_ DEFINITION session.

Format

HELP or
HEL
SUBJECT=string
MANUAL=file
STATUS=status_variable

Parameters

SUBJECT or S

Topic to be located in the online manual index. The topic must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands.

MANUAL or M

File containing the online manual whose index is searched. If you omit the MANUAL parameter, the default is AFM. The working catalog is searched for the file and then the $SYSTEM.MANUALS catalog.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- If you enter a topic that is not in the manual index, a message appears telling you that the topic could not be found.
- The default manual file, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for full-screen applications, the online manual is displayed in screen mode. To leave the online manual, press the QUIT function key.
- To request help in reading the online manual, enter HELP while in the manual.

QUIT Subcommand (for the CREATE_KEY_DEFINITION Utility)

Purpose Exits the Create_Key_Definition utility, ending specification of the concatenated-key.

Format

QUIT or
QUI
STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks Entry of the QUIT subcommand returns you to the CREATE_ALTERNATE_INDEXES utility session. This is indicated by the prompt creai/.

Examples This CREATE_ALTERNATE_INDEXES session defines a concatenated alternate key having two pieces. The first piece is the ten bytes beginning at byte 8. (Remember, bytes are numbered from the left beginning with zero.) The second piece is the eight-byte integer at the beginning of the record.

```
/create_alternate_indexes input=$user.is_file
creai/create_key_definition alternate_key_3 ..
creai../key_position=8 key_length=10 ..
creai../concatenated_pieces=yes
crekd/add_piece key_position=0 key_length=8 ..
crekd../key_type=integer
crekd/quit              "Exits CREATE_KEY_DEFINITIONS.
creai/quit no           "Exits CREATE_ALTERNATE_INDEXES
                        "without applying the
                        "alternate-key definition.
```

DELETE_KEY_DEFINITION Subcommand

Purpose Requests the deletion of an existing alternate key within a CREATE_ALTERNATE_INDEXES utility session.

Format

DELETE_KEY_DEFINITION or
DELKD
KEY_NAME=name
STATUS=status_variable

Parameters

KEY_NAME or NAME or KN or N

Name of the alternate key to be deleted. This parameter is required.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- The DELETE_KEY_DEFINITION subcommand requests deletion of an alternate key but does not actually delete the key from the file. The deletion remains pending until it is applied by an APPLY_KEY_DEFINITIONS or QUIT subcommand or it is canceled by a CANCEL_KEY_DEFINITIONS subcommand.

- You could use the following subcommand to list the alternate-key names:

```
display_key_definitions, all, display_option=brief
```

Examples This CREATE_ALTERNATE_INDEXES session deletes an alternate key named ALTERNATE_KEY_1.

```
/create_alternate_indexes input=$user.is_file
creai/delete_key_definition key_name=alternate_key_1
creai/quit apply_key_definitions=yes
-- File :NVE.USER99.IS_FILE : begin deleting alternate key definitions.
-- File :NVE.USER99.IS_FILE : alternate key ALTERNATE_KEY_1 has been
deleted.
-- File :NVE.USER99.IS_FILE : end deleting alternate key definitions.
```

DISPLAY_KEY_DEFINITIONS Subcommand

Purpose Displays alternate-key definitions within a CREATE_ALTERNATE_INDEXES session.

Format

DISPLAY_KEY_DEFINITIONS or
DISPLAY_KEY_DEFINITION or
DISKD
KEY_NAMES=keyword_value or list of names
DISPLAY_OPTIONS=keyword_value
SAMPLE_RECORD_COUNT=integer or keyword_value
OUTPUT=file_reference
STATUS=status_variable

Parameters

KEY_NAMES or KEY_NAME or NAMES or NAME or KN or N

Indicates the alternate-key definitions displayed.

list of names	Displays the specified alternate-key definitions.
PENDING or P	Displays only the pending alternate-key creations and deletions.
ALL or A	Displays both pending and existing alternate-key definitions.

If you omit the KEY_NAMES parameter, only the pending alternate-key creations and deletions are displayed.

DISPLAY_OPTIONS or DISPLAY_OPTION or DO

Indicates the contents of the display.

BRIEF or B	Displays the key name, position, length, type, and state.
FULL or F	Displays all information in the alternate-key definition.
SAMPLE_RECORDS or SR	Displays only sample records with the alternate keys marked.

BRIEF_SAMPLE_RECORDS or BSR	Displays the brief definition and the sample records.
FULL_SAMPLE_RECORDS or FSR or ALL or A	Displays the full definition and the sample records.

If you omit the DISPLAY_OPTIONS parameter, ALL is used (full definition and sample records).

SAMPLE_RECORD_COUNT or SRC

Indicates the number of records displayed if the DISPLAY_OPTIONS parameter requests a sample record display.

integer (0 through 4398046511103)	Displays the specified number of records.
ALL or A	Displays all records in the file.

The default is a one-record display.

OUTPUT or O

File to which the display is written. If you omit the OUTPUT parameter, the display is written to the standard file $OUTPUT.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- A sample-record display shows the record contents in ASCII characters and hexadecimal digits. For a fixed-length key, the alternate-key fields are underscored. For a variable-length key, the alternate-key values are underscored.

- The underscores for each alternate key appear on a separate line as follows:
  - If the concatenated-key or repeating-groups attributes are not defined for the key, the underscore characters indicate the alternate-key type (C for collated, I for integer, or U for uncollated).
  - If the key is a concatenated key, the underscores for each key field include one or two letters. The fields concatenated are a_, b_, and so forth up to z_ and then, aa, ba, ca, and so forth.
  - If the alternate-key definition specifies repeating groups, the underscores for each alternate-key value in the record include a number (1, 2, and so forth).

Examples This CREATE_ALTERNATE_INDEXES session writes a display to file LIST. The listing includes all records in the file, marked with the proposed alternate-key ALTERNATE_KEY_2.

```
/create_alternate_indexes  input=$user.is_file
creai/create_key_definition  key_name=alternate_key_2 ..
creai../key_position=0  key_length=2 ..
creai../repeating_group_length=20
creai/display_key_definitions  ..
creai../display_option=sample_records ..
creai../sample_record_count=all  output=list
creai/quit  apply_key_definitions=no
```

The following CREATE_ALTERNATE_INDEXES session contains a DISPLAY_KEY_DEFINITIONS subcommand for a default display, that is, a full definition of all pending alternate-key creations and deletions and a single sample record.

```
/create_alternate_indexes input=$user.is_file
creai/create_key_definition key_name=alternate_key_1 key_position=0 ..
creai../key_length=2 repeating_group_length=4
creai/display_key_definitions
              Display_Key_Definitions                    1985-10-03
NOS/VE Keyed File Utilities 1.1 85259                      14:09:01
File = .NVE.USER99.IS_FILE

 KEY NAME                          POSITION LENGTH TYPE       STATE
 --------------------------------- -------- ------ ---------- -----------
 ALTERNATE_KEY_1                          0      2 uncollated Creation
                                                              pending
   Duplicate_Key_Values             : not_allowed
   Null_Suppression                 : no
   Repeating_Groups_Specified
     Repeating_Group_Length         : 4
     Repeating_Group_Count          : repeat_to_end_of_record
================================================================================
RECORD 1 ........(in ascii) : T h i s   i s   t h e   f i r s t   r e c o r
                ( in hex ) : 546869732069732074686520666972737420726563 6F72
ALTERNATE_KEY_1             : 1_1_   2_2_   3_3_   4_4_   5_5_   6_6_
                    (in ascii) : d .
                    ( in hex ) : 642E
                                 >
creai/quit apply_key_definitions=no
```

HELP Subcommand (for the CREATE_ALTERNATE_INDEXES Utility)

Purpose Provides online help from within a CREATE_ALTERNATE_INDEXES session.

Format

HELP or
HEL
SUBJECT=string
MANUAL=file
STATUS=status_variable

Parameters

SUBJECT or S

Topic to be located in the online manual index. The topic must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands.

MANUAL or M

File containing the online manual whose index is searched. If you omit the MANUAL parameter, the default is AFM. The working catalog is searched for the file and then the $SYSTEM.MANUALS catalog.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- If you enter a topic that is not in the manual index, a message appears telling you that the topic could not be found.
- The default manual file, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for full-screen applications, the online manual is displayed in screen mode. To leave the online manual, press the QUIT function key.
- To request help in reading the online manual, enter HELP while in the manual.

QUIT Subcommand (for the CREATE_ALTERNATE_INDEXES Utility)

Purpose Ends the CREATE_ALTERNATE_INDEXES utility session.

Format

QUIT or
QUI
APPLY_KEY_DEFINITIONS = boolean or keyword_value
ERROR_LIMIT = integer
STATUS = status_variable

Parameters

APPLY_KEY_DEFINITIONS or APPLY_KEY_DEFINITION or AKD

Indicates how pending alternate-key creation and deletion requests are processed.

APPLY or A or TRUE or ON or YES	Apply all pending creation and deletion requests.
CANCEL or C or FALSE or OFF or NO	Cancel all pending creation and deletion requests.

This parameter is required if creation or deletion requests are pending.

ERROR_LIMIT or EL

Number of nonfatal (trivial) errors allowed for the apply operation (integer from 0 through 4398046511103 [2^{42} - 1]).

A 0 value indicates no limit; 0 is the default value.

See the APPLY_KEY_DEFINITIONS description for a description of apply error processing.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- The APPLY_KEY_DEFINITIONS parameter is required only if alternate-key creation or deletion requests are pending. In this case, you must specify whether to apply or cancel the pending requests.

- If you request application of the pending creations and deletions, the QUIT subcommand (before exiting the utility) performs the same processing as the APPLY_KEY_DEFINITIONS subcommand.

  Similarly, if you request cancellation of the requests, the QUIT subcommand performs the same processing as the CANCEL_KEY_DEFINITIONS subcommand before exiting the utility.

  For more information, see the APPLY_KEY_DEFINITIONS and CANCEL_KEY_DEFINITIONS subcommand descriptions.

Examples

This CHANGE_ALTERNATE_INDEXES session requests an alternate-key deletion and an alternate-key creation, but then cancels the requests:

```
/change_alternate_indexes file=$user.isfile
creai/delete_key_definition alternate_key_1
creai/create_key_definition alternate_key_1 ..
creai../key_position=0 key_length=5 key_type=integer
creai/quit apply_key_definitions=no
```

CREATE_KEYED_FILE and CHANGE_KEYED_FILE Utilities 8

CREATE_KEYED_FILE and CHANGE_KEYED_FILE Utilities 8

This chapter describes the use of two NOS/VE command utilities: CREATE_KEYED_FILE and CHANGE_KEYED_FILE. These utilities can manipulate nested file definitions and the records stored in keyed files.

The CREATE_KEYED_FILE command utility creates a new keyed file. (The keyed file must be previously defined by a SET_FILE_ATTRIBUTE command.) The RECOVER_KEYED_FILE/ommand utility can change an existing keyed file or a copy of an existing keyed file.

You can create and change alternate-key definitions while using CREATE_KEYED_FILE or CHANGE_KEYED_FILE by executing the Create_Alternate_Indexes utility as a subutility in the session. The subcommands for the CREATE_ALTERNATE_INDEXES subutility are described in detail in chapter 7.

Using the Utilities

The CREATE_KEYED_FILE and CHANGE_KEYED_FILE command utilities use the same subcommands. However, the two utilities differ in the keyed file used as follows:

- The file specified on the CREATE_KEYED_FILE command must not exist, but its attributes must be previously defined by a SET_FILE_ATTRIBUTES command.
- The input file specified on the CHANGE_KEYED_FILE command must be an existing keyed file. The command can also specify an output file which may not be an existing file. If an output file is specified, CHANGE_KEYED_FILE copies the input file to the output file; the session subcommands manipulate the contents of the output file.

Online help is available in a utility session via the HELP subcommand. A utility session ends when the QUIT subcommand is entered. The utility prompts are crekf/ and chakf/. Any NOS/VE command can be entered in response to these prompts.

Preparation Before Using the Utilities

Before using the CREATE_KEYED_FILE and CHANGE_KEYED_FILE utilities, you may need to enter certain NOS/VE commands as follows:

- Before creating a new keyed file, you must first define the keyed file with a SET_FILE_ATTRIBUTES command. The new file must be defined, but it must not be used for any other purpose (that is, opened) before the CREATE_KEYED_FILE session.

  The attributes used to define a new keyed file are described in chapter 6. The following is an example of a command to create an indexed-sequential file.

```
/set_file_attributes, file=$user.my_isfile, ..
../file_organization=indexed_sequential, record_type=fixed, ..
../maximum_record_length=80, key_length=10 ..
```

- Before creating or changing a keyed file that uses a user-defined hashing procedure, collation table, or compression procedure, you must add the object library containing the procedure or collation table to the program library list. The object library must be added before the utility session begins.

  For example, the following command adds an object library to the program library list.

```
/set_program_attribute, add_library=$user.my_hash_library
```

  For more information on the program library list and the commands that affect it, see the NOS/VE Object Code Management Usage manual.

Manipulating Nested Files

The CREATE_KEYED_FILE or CHANGE_KEYED_FILE session can manipulate the nested-file definitions in the file. One nested file (named $MAIN_FILE) is created when the file is created.

The following subcommands manipulate nested files:

Subcommands	Purpose
CREATE_NESTED_FILE	Creates and selects a new nested file.
DELETE_NESTED_FILE	Deletes one or more nested files.
DISPLAY_NESTED_FILE	Displays information about the nested files.
SELECT_NESTED_FILE	Selects the nested file to become the currently selected nested file.

The other utility subcommands reference one nested file, the currently selected nested file. A CREATE_ALTERNATE_INDEXES subutility session applies only to the currently selected nested file. Initially, the currently selected nested file is $MAIN_FILE. You can select another nested file with the SELECT_NESTED_FILE subcommand or create and select a new nested file with the CREATE_NESTED_FILE subcommand.

Adding and Replacing Records From Input Files

The following subcommands copy records from the input files specified on the subcommand:

Subcommands	Purpose
ADD_RECORDS	Puts the records into the selected nested file. (Their primary-key values must be unique in the nested file.)
COMBINE_RECORDS	Puts the records with new primary-key values and replaces the records with existing primary-key values.
REPLACE_RECORDS	Replaces records in the selected nested file. (Their primary-key values must already exist in the nested file.)

The following rules apply to the record manipulation subcommands:

- The subcommand operates only on the records in the currently selected nested file. The initially selected nested file is $MAIN_FILE. You can change the selected nested file with a SELECT_NESTED_FILE or CREATE_NESTED_FILE subcommand.
- When SORT=TRUE is specified, the input records for an indexed-sequential file are sorted by primary-key value; input records for a direct-access file are sorted by hash result value.

  For Better Performance

  Pre-sorting the input records by specifying SORT=TRUE or by using the SORT command is highly recommended. (The SORT command is described in chapter 2 of this manual.)

  When SORT=TRUE, the subcommand uses additional temporary space to sort the input records. This additional space could be significant if the input files are very large. If necessary, specify SORT=FALSE to prevent sorting of the input records.

  When SORT=TRUE, the subcommand writes any records with duplicate primary key values to the temporary file AAF$CREKF_DUPLICATE_LOG. The records written to AAF$CREKF_DUPLICATE_LOG are in no specific order.

- The ERROR_LIMIT file attribute controls the maximum number of nonfatal errors that can occur before the nonfatal errors cause a fatal error (error_limit_exceeded) that terminates the subcommand.

  If you receive a fatal error during subcommand processing, processing is suspended with records added only up to the record that caused the error. A message tells you how to add the rest of the records.

  To do so, you enter, in response to the crekf/ prompt, another subcommand specifying AAF$CONTINUE as the only input file. (AAF$CONTINUE is a temporary file in which the command has copied the input records. It is positioned at the record following the record in error.) The subcomand can also specify the ERROR_ LIMIT parameter.

- The subcommand detects a nonfatal error when a record:

  - Is shorter than the MINIMUM_RECORD_LENGTH or longer than the MAXIMUM_RECORD_LENGTH of the nested file.

  - Contains an alternate-key value that duplicates an existing key value.

  - Contains a duplicate primary-key value.

  - Contains a sparse-key value that indicates that the alternate-key value should be included in the alternate index, but the record is too short to include the alternate-key field.

- If the output file has fixed-length (F) records and the input record is shorter than the MAXIMUM_RECORD_LENGTH of the output file, the subcommand pads the output record to the maximum record length using the PADDING_CHARACTER attribute value. The default padding character is the space.

- If you specify the standard input file $INPUT on the subcommand in an interactive session, you are prompted for record entry from the terminal. (The prompt is the PROMPT_STRING connection attribute value. Unless changed, its value is ?.)

  To end record entry, enter the END_OF_INFORMATION connection attribute value as the response to the prompt. Unless changed, its value is *EOI.

  For example:

```
crekf/add_records, $input
? This is a new record to be added.
? *EOI
crekf/
```

Selecting Records

The following subcommands manipulate selected records in the currently selected nested file:

Subcommands	Purpose
DELETE_RECORDS	Deletes selected records from the nested file.
DISPLAY_RECORDS	Formats and displays selected records.
EXTRACT_RECORDS	Copies selected records from the nested file to another file.

The following rules apply to the subcommands that select records:

- The subcommand only selects records from the currently selected nested file.
- When specifying values for a key range, you do not have to specify the entire key value. For example, if you specify KEYS='A'..'Z', the range begins with the first key value whose first character is greater than or equal to A and ends at the first key value whose first character is greater than Z.
- If you specify the KEYS parameter with only one key value, the full key value must be specified. This is to ensure that the correct record is selected. (This rule is not effective for a DELETE_RECORDS subcommand with VETO=TRUE, because in that case, you must confirm that the correct record is being deleted.)
- When you specify a single key value on the KEY parameter, the subcommand searches for that key value and, if it cannot find that value, it does not select any records. In contrast, when you specify a range on the KEY parameter, the subcommand searches for that key value or the next greater key value.

For example, suppose a file has only two records; their primary-key values are B and C. The following subcommands are entered:

delete_record, key='A'
No record has key value A so no records are selected.
delete_record, key='A'..'2'
Both records are selected.
delete_record, key='A', count=20
No record has key value A so no records are selected.
delete_record, key='A'..$LAST_KEY, count=20
Both records are selected.

- The COUNT parameter selects records until either the count is satisfied or the upper value of the key range is encountered.

Calculating the INITIAL_HOME_BLOCK_COUNT

Creation of a new keyed file or nested file using the direct-access file organization requires an INITIAL_HOME_BLOCK_COUNT value for the file. (The INITIAL_HOME_BLOCK_COUNT value is the number of blocks allocated for the file when it is created, as described under Direct-Access File Structure in chapter 5.)

For a new keyed file, you can specify the INITIAL_HOME_BLOCK_COUNT value on the SET_FILE_ATTRIBUTES command for the file or have the CREATE_KEYED_FILE command calculate the value for you. Similarly, for a new nested file, you can specify the INITIAL_BLOCK_COUNT on the CREATE_NESTED_FILE subcommand or have the subcommand calculate the value for you.

In both cases, if you omit the INITIAL_HOME_BLOCK_COUNT value, you receive the following message and the prompt IHBC/:

> --INFORMATIVE-- The output file is a direct-access file but a value was not provided for INITIAL_HOME_BLOCK_COUNT. Please enter ADD_RECORDS commands in response to the prompt "IHBC/". Enter QUIT after specifying ALL files to be copied. CREATE_KEYED_FILE will then compute the INITIAL_HOME_BLOCK_COUNT and copy the records into the new file. IHBC/

The command should calculate the INITIAL_HOME_BLOCK_COUNT value only if all input records for the new keyed file or new nested file are available. If additional records are to be added later, after the creation of the file or nested file, you should enter QUIT after the IHBC/ prompt. In this case, you must choose an INITIAL_HOME_BLOCK_COUNT for the file or nested file and specify it on the command or subcommand.

However, if all records to be stored in the direct-access file are available, you can have the command or subcommand calculate the appropriate initial_home_block_count for you. To have the initial_home_block_count calculated for you, you must specify the files containing ALL input records for the new file or nested file. You specify the files on ADD_RECORDS subcommands.

The command or subcommand reads the files specified on the ADD_RECORDS subcommand and accumulates the records to be put into the file. When you enter the QUIT subcommand, it calculates the initial_home_block_count value, creates the direct-access file, and puts the records into the file.

The initial_home_block_count is calculated using this formula:

```
(average_record_length + 3) * record_count
-------------------------------------------
(loading_factor/100) * (block_length - 36)
```

You can specify the loading factor used by the formula on the CREATE_KEYED_FILE command or CREATE_NESTED_FILE subcommand. The default is 75.

For example, the following commands create a direct-access file containing the records from a file and a nested file:

```
IHBC/add_records, $user.sequential_file
IHBC/add_records, ($user.keyed_file, $main_file)
IHBC/quit
--INFORMATIVE--  INITIAL_HOME_BLOCK_COUNT=105 based upon
LOAD_FACTOR=75, AVERAGE_RECORD_LENGTH=63,
RECORD_COUNT=12000 and BLOCK_LENGTH=512.
crekf/
```

CREATE_KEYED_FILE Example

The following interactive session illustrates use of the CREATE_KEYED_FILE command utility:

```
/copy_keyed_file input=$user.add_file        "Displays contents of
Everest         Africa          8800         "$USER.ADD_FILE.
Fuji            Asia            7000
K2              Asia            8611
Kilimanjaro     Africa          5895

/copy_keyed_file input=$user.replace_file    "Displays contents of
Everest         Asia            8848         "$USER.REPLACE_FILE.

/copy_keyed_file input=$user.combine_file    "Displays contents of
Matterhorn      Europe          4478         "$USER.COMBINE_FILE.
McKinley        North America   6194
Fuji            Asia            6999

/set_file_attributes ..                 "Defines the file attributes
../file=$user.indexed_sequential_file ..  "of file $USER.INDEXED_
../file_organization=indexed_sequential ..  "SEQUENTIAL_FILE.
../maximum_record_length=32 ..
../minimum_record_length=14 ..
../key_length=14, key_position=0

/create_keyed_file ..                   "Starts the utility session.
../output=$user.indexed_sequential_file

crekf/add_records ..                             "Adds records.
crekf../input=$user.add_file

crekf/replace_records ..                         "Replaces records.
crekf../input=$user.replace_file

crekf/combine_records ..                  "Adds and replaces records.
crekf../input=$user.combine_file

crekf/display_records count=all                  "Displays records.
                Display_Nested_File                          1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                        11:19:36
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
```

```
 Byte: 0              ASCII: Everest       Asia          8848
 Byte: 0              ASCII: Fuji          Asia          6999
 Byte: 0              ASCII: K2            Asia          8611
 Byte: 0              ASCII: Kilimanjaro   Africa        5895
 Byte: 0              ASCII: Matterhorn    Europe        4478
 Byte: 0              ASCII: McKinley      North America 6194

crekf/extract_records output=new_file ..    "Extracts records.
crekf../keys='E'..'Ma', count=3
--INFORMATIVE AA 501275-- The Extract_Records subcommand of
CREATE_KEYED_FILE copied 3 records from nested file
$MAIN_FILE in file :NVE.USER99.INDEXED_SEQUENTIAL_FILE
to NEW_FILE.

crekf/delete_records ..                      "Deletes records.
crekf../keys='Matterhorn'..'McKinley' ..
crekf../count=2, veto=true
Byte: 0              ASCII: Matterhorn     Europe        4478
Okay to delete?  ==>Yes
Byte: 0              ASCII: McKinley       North America 6194
Okay to delete?  ==>No
--INFORMATIVE AA 501305--  As requested by the user, this
record was not deleted.
--INFORMATIVE AA 501285--  The Delete_Records subcommand of
CREATE_KEYED_FILE deleted 1 record from nested file
$MAIN_FILE in file :NVE.USER99.INDEXED_SEQUENTIAL_FILE.

crekf/create_nested_file ..        "Creates a new nested file.
crekf../name=nested_file_1 ..
crekf../maximum_record_length=32 ..
crekf../key_length=14 ..
crekf../file_organization=indexed_sequential

crekf/display_nested_file          "Displays the nested files.
           Display_Nested_File                        1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                  12:20:36
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    NESTED_FILE_1            (currently selected nested file)
    $MAIN_FILE

crekf/select_nested_file ..      "Selects another nested file.
crekf../name=$main_file
```

```
crekf/display_nested_file          "Displays the nested files.
              Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                 12:25:36
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    $MAIN_FILE                (currently selected nested file)
    NESTED_FILE_1

crekf/delete_nested_file ..            "Deletes a nested file.
crekf../name=nested_file_1

crekf/display_nested_file          "Displays the nested files.
              Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                 12:30:46
File = :NVE.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    $MAIN_FILE                (currently selected nested file)

crekf/quit                                         "Ends the session.
```

CREATE_KEYED_FILE Command

Purpose Creates the keyed file specified on the command and begins a CREATE_KEYED_FILE utility session.

NOTE

The new keyed file must be previously defined by a SET_FILE_ATTRIBUTES command.

Format

CREATE_KEYED_FILE or
CREKF
OUTPUT=file
STATUS=status_variable

Parameters

OUTPUT or OB

File path of the keyed file to be created. The file must be a new file (never opened), but its attributes must have been specified by previous SET_FILE_ATTRIBUTES commands. This parameter is required.

The minimum attributes that must be defined are KEY_LENGTH and MAXIMUM_RECORD_LENGTH. If the FILE_ORGANIZATION is omitted, CREATE_KEYED_FILE creates an indexed-sequential file.

STATUS

Optional status variable.

In a batch job, when a subcommand without its own status variable returns an error, the error is stored in the CREATE_KEYED_FILE status variable, if any, and the utility session terminates.

In an interactive session, when a subcommand without its own status variable returns an error, the error is displayed at the terminal, but the utility session does not end. The user can continue the session with another subcommand.

In an interactive session, only the completion status of the CREATE_KEYED_FILE command and the QUIT command that ends the session are stored in the CREATE_KEYED_FILE status variable.

Remarks

- The command utility prompt is:

  ```
  crekf/
  ```

  In response to the crekf/ prompt, you can enter NOS/VE commands and any of these subcommands:

  ADD_RECORDS
  COMBINE_RECORDS
  CREATE_ALTERNATE_INDEXES
  CREATE_NESTED_FILE
  DELETE_NESTED_FILE
  DELETE_RECORDS
  DISPLAY_NESTED_FILE
  DISPLAY_RECORDS
  EXTRACT_RECORDS
  HELP
  QUIT
  REPLACE_RECORDS
  SELECT_NESTED_FILE

- The new keyed file is created with one nested file, named $MAIN_FILE. It is the initially selected nested file and all subcommands apply to it until a CREATE_NESTED_FILE or SELECT_NESTED_FILE subcommand selects another nested file.

- If any nested file in the new keyed file will use a user-defined collation table, hashing procedure, or compression procedure, the object library containing the compiled table or procedure must be in the program library list before the CREATE_KEYED_FILE session begins.

  To add one or more object libraries to the program library list, use the ADD_LIBRARIES parameter on a SET_PROGRAM_ATTRIBUTES command. For example:

  ```
  set_program_attributes, add_library=$user.hash_library
  ```

- If you specify DIRECT_ACCESS as the FILE_ORGANIZATION attribute on the SET_FILE_ATTRIBUTES command, but omit the INITIAL_HOME_BLOCK_COUNT attribute, CREATE_KEYED_FILE prompts you for calculation of the INITIAL_HOME_BLOCK_COUNT. For more information, see Calculating the INITIAL_HOME_BLOCK_COUNT earlier in this chapter.

Examples

This CREATE_KEYED_FILE example defines the file $USER.INDEXED_SEQUENTIAL_FILE with the SET_FILE_ATTRIBUTES command and then creates it.

```
/set_file_attributes..
../file=$user.indexed_sequential_file ..
../file_organization=indexed_sequential ..
../maximum_record_length=32..
../minimum_record_length=14 ..
../key_length=14
/create_keyed_file..
../output=$user.indexed_sequential_file
crekf/
```

At this point, the file has been opened and, therefore, exists, but is empty.

CHANGE_KEYED_FILE Command

Purpose Changes an existing keyed file or a copy of the keyed file and begins a CHANGE_KEYED_FILE utility session.

Format

CHANGE_KEYED_FILE or
CHAKF
INPUT=file
OUTPUT=file
STATUS=status_variable

Parameters

INPUT or I

File path of an existing keyed file. You must have at least read access to the input file. This parameter is required.

OUTPUT or O

File path of the keyed file to which the input file is copied.

If file does not exist, CHANGE_KEYED_FILE creates it when it copies the input file. If the file does exist, it must have the same attributes as the input file.

If you omit the OUTPUT parameter, CHANGE_KEYED_FILE does not use an output file; instead, it opens the input file and changes it directly.

STATUS

Optional status variable.

In a batch job, when a subcommand without its own status variable returns an error, the error is stored in the CHANGE_KEYED_FILE status variable, if any, and the utility session terminates.

In an interactive session, when a subcommand without its own status variable returns an error, the error is displayed at the terminal, but the utility session does not end. The user can continue the session with another subcommand.

In an interactive session, only the completion status of the CHANGE_KEYED_FILE command and the QUIT command that ends the session are stored in the CHANGE_KEYED_FILE status variable.

Remarks

- The command utility prompt is:

```
chakf/
```

- In response to the chakf/ prompt, you can enter NOS/VE commands and any of these subcommands:

  ADD_RECORDS
  COMBINE_RECORDS
  CREATE_ALTERNATE_INDEXES
  CREATE_NESTED_FILE
  DELETE_NESTED_FILE
  DELETE_RECORDS
  DISPLAY_NESTED_FILE
  DISPLAY_RECORDS
  EXTRACT_RECORDS
  HELP
  QUIT
  REPLACE_RECORDS
  SELECT_NESTED_FILE

- All subcommands in the session apply to the currently selected nested file. The initially selected nested file is $MAIN_FILE. The nested file selection can be changed by a CREATE_NESTED_FILE or SELECT_NESTED_FILE subcommand.

- If the existing keyed file or a new nested file to be created uses a user-defined collation table, hashing procedure, or compression procedure, the object library containing the compiled table or procedure must be in the program library list before the CHANGE_KEYED_FILE session begins.

  To add one or more object libraries to the program library list, use the ADD_LIBRARIES parameter on a SET_PROGRAM_ATTRIBUTES command. For example:

```
set_program_attributes, add_library=$user.hash_library
```

Examples

The following session copies an existing keyed file and then ends.

```
/change_keyed_file, input=$user.existing_keyed_file, ..
../output=$user.new_keyed_file
chakf/quit
```

ADD_RECORDS Subcommand

Purpose Puts records into the currently selected nested file.

Format

ADD_RECORDS or
ADD_RECORD or
ADDR
INPUT=file or list of files
SORT=boolean
ERROR_LIMIT=integer
STATUS=status_variable

Parameters

INPUT or I

List of one or more files whose records are to be copied. You must have at least read access to the input files. This parameter is required.

SORT or S

Indicates whether the subcommand sorts the input records before adding them to the currently selected nested file. If you omit the SORT parameter, the default is SORT=TRUE.

ERROR_LIMIT or EL

Number of nonfatal (trivial) errors allowed for the ADD_RECORDS operation (integer from 0 through 4398046511103[2^{42}-1]).

A 0 value indicates no limit; 0 is the default value.

See the Remarks for a description of error processing.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks See the rules for record copying subcommands listed under Adding and Replacing Records From Input Files at the beginning of this chapter.

Examples

This CREATE_KEYED_FILE example creates the file $USER.INDEXED_SEQUENTIAL_FILE, adds the records of file $USER.ADD_RECORDS to it, and then displays the file.

```
/set_file_attributes ..
../file=$user.indexed_sequential_file ..
../file_organization=indexed_sequential ..
../maximum_record_length=32 ..
../minimum_record_length=14 ..
../key_length=14

/create_keyed_file ..
../output=$user.indexed_sequential_file
crekf/add_records input=$user.add_records

crekf/display_records count=all

Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                    11:19:36
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
Display of records in $MAIN_FILE

 Byte: 0            ASCII: Everest       Asia          8848
 Byte: 0            ASCII: K2            Asia          8611
 Byte: 0            ASCII: Kilimanjaro   Africa        5895
 Byte: 0            ASCII: Matterhorn    Europe        4478
 Byte: 0            ASCII: McKinley      North America 6194
crekf/
```

COMBINE_RECORDS Subcommand

Purpose Puts and replaces records in the currently selected nested file.

Format

COMBINE_RECORDS or
COMBINE_RECORD or
COMR
INPUT=file or list of files
SORT=boolean
ERROR_LIMIT=integer
STATUS=status_variable

Parameters

INPUT or I

List of one or more files whose records are to be copied. You must have at least read permission to the input files. This parameter is required.

SORT or S

Indicates whether the subcommand sorts the input records combining them with the currently selected nested file. If you omit the SORT parameter, the default is SORT=TRUE.

ERROR_LIMIT or EL

Number of nonfatal (trivial) errors allowed for the COMBINE_RECORDS operation (integer from 0 through 4398046511103[2^{42}-1]).

A 0 value indicates no limit; 0 is the default value.

See the Remarks for a description of error processing.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- Because the COMBINE_RECORDS subcommand does not lock key values it is recommended that you do not use this subcommand when another task could be updating the same records. If you attach the output file with no write share modes (SHARE_MODES=READ or SHARE_MODES=NONE) before the session, the file cannot be shared for updating.

- See the rules for record copying subcommands listed under Adding and Replacing Records From Input Files at the beginning of this chapter.

Examples This CREATE_KEYED_FILE example adds records that have a new primary key and replaces records that have an existing primary-key value.

```
/copy_keyed_file add_file
Everest       Africa        8800
K2            Asia          8611
Kilimanjaro   Africa        5895

/copy_keyed_file combine_file
Everest       Asia          8848
Matterhorn    Europe        4478
McKinley      North America 6194

/create_keyed_file ..
../output=$user.indexed_sequential_file
crekf/add_records input=$user.add_file
crekf/combine_records input=$user.combine_file
crekf/display_records count=all
                Display_Nested_File                          1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                        12:01:46
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
Display of records in $MAIN_FILE

 Byte: 0              ASCII: Everest       Asia          8848
 Byte: 0              ASCII: K2            Asia          8611
 Byte: 0              ASCII: Kilimanjaro   Africa        5895
 Byte: 0              ASCII: Matterhorn    Europe        4478
 Byte: 0              ASCII: McKinley      North America 6194
crekf/
```

CREATE_ALTERNATE_INDEXES Subcommand

Purpose Begins a CREATE_ALTERNATE_INDEXES subutility session to create, delete, and display alternate-key definitions in the currently selected nested file.

Format

CREATE_ALTERNATE_INDEXES or
CHANGE_ALTERNATE_INDEXES or
CREATE_ALTERNATE_INDICES or
CHANGE_ALTERNATE_INDICES or
CREATE_ALTERNATE_INDEX or
CHANGE_ALTERNATE_INDEX or
CREAI or
CHAAI
STATUS=status_variable

Parameters *STATUS*

Optional status variable.

In a batch job, when a subcommand without its own status variable returns an error, the error is stored in the CREATE_ALTERNATE_INDEXES status variable, if any, and the subutility session terminates.

In an interactive session, when a subcommand without its own status variable returns an error, the error is displayed at the terminal, but the subutility session does not end. The user can continue the session with another subcommand.

In an interactive session, only the completion status of the CREATE_ALTERNATE_INDEXES command and the QUIT command that ends the session are stored in the CREATE_ALTERNATE_INDEXES status variable.

Remarks

- The subutility prompt is:

  ```
  creai/
  ```

  In response to the creai/ prompt, you can enter SCL commands and any of these subcommands:

  ```
  CREATE_KEY_DEFINITIONS
  DISPLAY_KEY_DEFINITIONS
  DELETE_KEY_DEFINITIONS
  CANCEL_KEY_DEFINITIONS
  APPLY_KEY_DEFINITIONS
  HELP
  QUIT
  ```

  For detailed descriptions of the CREATE_ALTERNATE_INDEXES subcommands, see chapter 7.

- The CREATE_ALTERNATE_INDEXES subcommand does not check your file permissions; each subcommand you enter in the subutility session check that you have the required permissions to do the operation.

  To display key definitions, you must have at least read permission; to create, delete, cancel, or apply key definitions, you must have at least the three permissions: append, modify, and shorten.

Examples The following subutility session creates an alternate-key definition and then displays it.

```
crekf/create_alternate_indexes
creai/create_key_definitions ..
creai../key_name=alternate_key_1 ..
creai../key_position=28 key_length=4  .
creai/display_key_definitions display_options=all
Display_Nested_File                        1986-02-17
NOS/VE Keyed File Utilities  1.2 86034                    12:20:26
File = :NVE.INDEXED_SEQUENTIAL_FILE
Nested_File_Name

 KEY_NAME                        POSITION LENGTH   TYPE         STATE
------------------------- -------- -------- ----------- ---------------
ALTERNATE_KEY_1                        28      4   uncollated  creation
pending
 Duplicate_Key_Value              : not_allowed
 Null_Suppression                 : no
================================================================================

RECORD 1 ..(in ascii) :E v e r e s t                A s i a
          ( in hex ) :4576657265737420202020202020417369612020202020

ALTERNATE_KEY_1                :

                   (in ascii) :        8 8 4 8
                   ( in hex ) :2020202038383438
                              >        U_U_U_U_
creai/
```

CREATE_NESTED_FILE Subcommand

Purpose Creates and selects a new nested file.

Format

CREATE_NESTED_FILE or
CRENF
NAME=name
MAXIMUM_RECORD_LENGTH=integer
KEY_LENGTH=integer
KEY_POSITION=integer
KEY_TYPE=keyword
FILE_ORGANIZATION=keyword
EMBEDDED_KEY=boolean
MINIMUM_RECORD_LENGTH=integer
RECORD_TYPE=keyword
COMPRESSION_PROCEDURE_NAME=name
COLLATE_TABLE_NAME=name
DATA_PADDING=integer
INDEX_PADDING=integer
INITIAL_HOME_BLOCK_COUNT=integer
HASHING_PROCEDURE_NAME=name
DYNAMIC_HOME_BLOCK_SPACE=boolean
LOADING_FACTOR=integer
RECORDS_PER_BLOCK=integer
STATUS=status_variable

Parameters

NAME or N

Name of the new nested file. It must be unique in the keyed file. This parameter is required.

MAXIMUM_RECORD_LENGTH or MAXRL

Maximum number of bytes in a data record (from 1 through 65497). This parameter is required.

KEY_LENGTH or KL

Integer specifying the primary-key length in bytes (for integer keys, from 1 through 8; for other key types, from 1 through 255). This parameter is required.

KEY_POSITION or KP

Position of the leftmost byte in the primary key (specified only if the key is embedded). The byte positions in a record are numbered from the left, beginning with 0. The default is 0.

KEY_TYPE or KT

Primary key type: UNCOLLATED (UC), INTEGER (I), or COLLATED (C). The default is UNCOLLATED.

For a direct-access file, any value specified for the KEY_TYPE attribute is ignored; the KEY_TYPE attribute value for direct-access files is always UNCOLLATED.

FILE_ORGANIZATION or FO

Organization of the file: either INDEXED_SEQUENTIAL (IS) or DIRECT_ACCESS (DA). The default file organization is INDEXED_SEQUENTIAL.

EMBEDDED_KEY or EK

SCL boolean value indicating whether the primary key is part of the record data (embedded) or separate from the record data (nonembedded). The default is TRUE (embedded keys).

MINIMUM_RECORD_LENGTH or MINRL

Minimum number of bytes in a data record (from 0 through 65497).

If the RECORD_TYPE is FIXED, the default minimum record length is 0, but the length of each fixed-length record must be the MAXIMUM_RECORD_LENGTH value.

If the RECORD_TYPE is UNDEFINED or VARIABLE and the key is embedded, the default is the sum of the KEY_POSITION and KEY_LENGTH values. Otherwise, the default is 1.

For variable-length records, you should explicitly specify this attribute. The minimum record length must include:

- The primary-key field
- All fixed-length alternate-key fields (or their sparse-key control characters) unless the key repeats to the end of the record.

RECORD_TYPE or RT

Record type: FIXED (F), VARIABLE (V), or UNDEFINED (U).

For keyed files, the record types VARIABLE and UNDEFINED are processed as the same and the TRAILING_CHARACTER_DELIMITED (T) record type is not supported.)

COMPRESSION_PROCEDURE_NAME or CPN

Name of the compression procedure to be executed with this file. The compression procedure provided by the system is AMP$RECORD_COMPRESSION. A compression procedure is optional.

COLLATE_TABLE_NAME or CTN

Name of the collating sequence by which collated keys are ordered (required if the KEY_TYPE is COLLATED). This parameter may be specified for indexed-sequential nested files only.

The name can be the name of a NOS/VE predefined collating sequence or a user-defined collating sequence (an entry point in an object library). See appendix E for more information.

DATA_PADDING or DP

Percentage of data-block space left empty when a block is created (integer from 0 through 99). The default is 0.

The percentage must allow for storage of at least one maximum-length record per block. This parameter may be specified for indexed-sequential nested files only.

INDEX_PADDING or IP

Percentage of index-block space left empty when a block is created (integer from 0 through 99). The default is 0.

The percentage must allow for storage of at least three index records per block. (The index record length is the key length plus 4). This parameter may be specified for indexed-sequential nested files only.

INITIAL_HOME_BLOCK_COUNT or IHBC

Number of home blocks to be created in the file (1 through 2^{31}-1). This parameter may be specified for direct-access nested files only. For more information, see the Direct-Access File Structure discussion in chapter 5.

HASHING_PROCEDURE_NAME or HPN

Name of the hashing procedure to be executed with this file. The default hashing procedure is the one provided by the system AMP$SYSTEM_HASHING_PROCEDURE. This parameter may be specified for direct-access nested files only. For more information, see the hashing procedure discussion in chapter 5.

DYNAMIC_HOME_BLOCK_SPACE or DHBS

This parameter is reserved for future use. The parameter default is FALSE.

LOADING_FACTOR or LF

Percentage of home block space used in a direct-access file.

The default percentage is 75%. To allow for less-than-uniform distribution of records in the home blocks, the loading factor should be no greater than 90%. If the INITIAL HOME BLOCK_COUNT is specified, the loading factor is ignored.

See the Remarks section for more information on computing the INITIAL_HOME_BLOCK_COUNT.

RECORDS_PER_BLOCK or RPB

This parameter is reserved for future use.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- If the new nested file is to use a user-defined collation table, hashing procedure, or compression procedure, the object library containing the compiled table or procedure must be in the program library list before the CREATE_KEYED_FILE or CHANGE_KEYED_FILE session begins.

To add one or more object libraries to the program library list, use the ADD_LIBRARIES parameter on a SET_PROGRAM_ATTRIBUTES command. For example:

```
set_program_attributes, add_library=$user.hash_library
```

- If you specify DIRECT_ACCESS as the FILE_ORGANIZATION attribute on the CREATE_NESTED_FILE subcommand, but omit the INITIAL_HOME_BLOCK_COUNT attribute, you are prompted for calculation of the INITIAL_HOME_BLOCK_COUNT. For more information, see Calculating the INITIAL_HOME_BLOCK_COUNT earlier in this chapter.

Examples This CREATE_KEYED_FILE example creates a new nested file NESTED_FILE_1 and then displays the newly created nested file.

```
crekf/create_nested_file name=nested_file_1 ..
crefk../maximum_record_length=32, key_length=14 ..
crekf../file_organization=indexed_sequential
crekf/display_nested_file
                Display_Nested_File                          1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                        12:42:49
File = :NVE.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    NESTED_FILE_1                (currently selected nested file)
    $MAIN_FILE
```

DELETE_NESTED_FILE Subcommand

Purpose Deletes one or more nested files.

Format

DELETE_NESTED_FILE or
DELNF
NAME=nameor list of names
STATUS=status_variable

Parameters

NAME or N

List of one or more nested files to be deleted. This parameter is required.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- You cannot delete the currently selected nested file or $MAIN_FILE.
- To delete the currently selected nested file, select another nested file first using the SELECT_NESTED_FILE subcommand and then issue the DELETE_NESTED_FILE subcommand.
- To display the names of the nested files, enter a DISPLAY_NESTED_FILE subcommand.

Examples This CREATE_KEYED_FILE example displays the list of nested files and then deletes the nested file NESTED_FILE_2.

```
crekf/display_nested_file
                Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                   12:50:12
File = :NVE.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    NESTED_FILE_1              (currently selected nested file)
    NESTED_FILE_2
    $MAIN_FILE

crekf/delete_nested_file name=nested_file_2
crekf/display_nested_file
                Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                   12:52:02
File = :NVE.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    $MAIN_FILE                 (currently selected nested file)
    NESTED_FILE_1
```

DELETE_RECORDS Subcommand

Purpose Deletes records from the currently selected nested file.

Format

DELETE_RECORDS or
DELETE_RECORD or
DELR
KEYS=range of key
COUNT=integer
VETO=boolean
STATUS=status_variable

Parameters

KEYS or KEY or K

Optional range of primary-key values to be deleted. The range may be specified in one of the following ways:

- As two primary-key values separated by two periods (..). The first value must be lower than or equal to the second value.
- As one primary-key value specifying the beginning of the range. The number of records deleted is specified by the COUNT parameter.

For indexed-sequential files, you may use the keywords $FIRST_KEY and $LAST_KEY in the key range to specify the lowest and highest key values, respectively.

For direct-access files, you can specify only one key value; a range of key values is not permitted. The only exception to this is the range $FIRST_KEY..$LAST_KEY, which deletes all records in the currently selected nested file.

If you omit the KEYS parameter, no default is assumed and records are deleted beginning with the first record.

COUNT or C

Indicates the number of records to be deleted.

Integer (0 through 4398046511103)	Deletes the specified number of records.
ALL or A	Deletes all records in the file.

If you omit the COUNT parameter, the default is 1, unless a range is specified by the KEYS parameter and then the default is the number of records in the range. If both a key range and a count is specified, the count limits the number of records deleted within the range.

VETO or V

Indicates whether the interactive user must confirm each deletion.

TRUE or YES or ON	Confirmation requested.
FALSE or NO or OFF	No confirmation requested.

DELETE_RECORDS requests confirmation by displaying the record to be deleted following by the prompt, Okay to delete? ==>. You respond with one of the following:

YES or Y	Delete the record.
NO or N	Do not delete record.
QUIT or Q	Terminate the subcommand without deleting any more records.
HEX or H	Redisplays the current record using hexadecimal representation and reissues the prompt.
All or A	Continue deleting records without further confirmation prompts.

If this parameter is omitted, the default is FALSE.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- Because the DELETE_RECORDS subcommand does not lock key values, it is recommended that you do not use this subcommand when another task could be updating the same records. If you attach the output file with no write share modes (SHARE_MODES=READ or SHARE_MODES=NONE) before the session, the file cannot be shared for updating.
- See the rules listed under Selecting Records at the beginning of this chapter.

Examples This CREATE_KEYED_FILE example deletes a record in the currently selected nested file.

```
crekf/delete_records, keys='Matterhorn'..'McKinley' ..
crekf../count=2, veto=true
Byte: 0        ASCII: Matterhorn    Europe        4478
Okay to delete? ==>Yes
Byte: 0        ASCII: McKinley      North America  6194
Okay to delete? ==>No
--INFORMATIVE AA 501305--  As requested by the user, this record was
not deleted.
--INFORMATIVE AA 501285--  The Delete_Records subcommand of
CREATE_KEYED_FILE deleted 1 records from nested file $MAIN_FILE in
file :NVE.INDEXED_SEQUENTIAL_FILE.
crekf/
```

DISPLAY_NESTED_FILE Subcommand

Purpose Displays the following information for the specified nested files:

- Nested-file definition
- Names of its alternate keys
- Number of data records stored in the nested file

Format

DISPLAY_NESTED_FILE or
DISNF
NAME=name or list of names
OUTPUT=file
DISPLAY_OPTIONS=list of keywords
STATUS=status_ variable

Parameters

NAME or N

List of nested file names for which information is to be displayed. The default keyword ALL specifies all nested files in the keyed file.

OUTPUT or O

File to which the display is written. The file must be a sequential file. If you omit the OUTPUT parameter, the display is written to file $OUTPUT.

DISPLAY_OPTIONS or DO

List of keywords indicating the type of information to be displayed.

DEFINITIONS or DEFINITION or D	Nested file definition
KEY_NAMES or KEY_NAME or K	Alternate-key names for each nested file.
NAMES or NAME or N	Nested file names only.
RECORD_COUNTS or RECORD_COUNT or RC	Number of data records in each nested file.
ALL or A	All of the above.

If you omit the DISPLAY_OPTIONS parameter, the display lists only the nested-file names (NAMES).

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- The currently selected nested file is marked as such in the list of nested files.

Examples

This CREATE_KEYED_FILE example displays the default nested file ($MAIN_FILE) with the DISPLAY_OPTIONS parameter set to ALL. No alternate keys have been defined.

```
crekf/display_nested_file display_options=all
                Display_Nested_File                          1986-02-17
NOS/VE Keyed File Utilities  1.2 86034                        12:59:58
File = :NVE.INDEXED_SEQUENTIAL_FILE

$MAIN_FILE                      (currently selected nested file)
Record_Count                    : 3

Nested_File_Definitions
  Compression_Procedure_Name    : none
  Embedded_Key                  : yes
  Key_Position                  : 0
  Key_Length                    : 14
  Maximum_Record_Length         : 32
  Minimum_Record_Length         : 32
  Record_Type                   : undefined
  File_Organization             : indexed_sequential
  Key_Type                      : uncollated
  Collate_Table_Name            :
  Data_Padding                  : 0
  Index Padding                 : 0
```

DISPLAY_RECORDS Subcommand

Purpose Formats and displays the records in the currently selected nested file.

Format

DISPLAY_RECORDS or
DISPLAY_RECORD or
DISR

OUTPUT=file reference
KEYS=range of key
DISPLAY_OPTIONS=keyword_value
COUNT=integer
STATUS=status_variable

Parameters

OUTPUT or O

File to which the display is written. The file must be a sequential file. You must have at least append permission to the file. If you omit the OUTPUT parameter, records are written to the standard file $OUTPUT.

KEYS or KEY or K

Optional range of primary-key values to be displayed. The range may be specified in one of the following ways:

- As two primary-key values separated by two periods (..). The first value must be lower than or equal to the second value.
- As one primary-key value specifying the beginning of the range. The number of records displayed is specified by the COUNT parameter.

For indexed-sequential files, you may use the keywords $FIRST_KEY and $LAST_KEY in the key range to specify the lowest and highest key values, respectively.

For direct-access files, you can specify only one key value; a range of key values is not permitted.

If you omit the KEYS parameter, no default is assumed and records are displayed beginning with the first record.

DISPLAY_OPTIONS or DISPLAY_OPTION or DO

List of one or more keyword values indicating the representation used for the contents of records.

ASCII	ASCII characters.
HEX or H	Hexadecimal digits.
BOTH	Both ASCII characters and hexadecimal digits.
ALTERNATE_KEY_DEFINITION or AKD or ALL	Records are displayed in both ASCII and hexadecimal with the alternate-key values marked.

If you omit the DISPLAY_OPTION parameter, the representation used is ASCII.

COUNT or C

Indicates the number of records to be displayed.

Integer (0 through 4398046511103)	Displays the specified number of records.
ALL or A	Displays all records in the file.

If you omit the COUNT parameter, the default is 1, unless a range is specified by the KEYS parameter and then it is the number of records in the range. If both a key range and a count are specified, the count limits the number of records displayed in the range.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- See the rules listed under Selecting Records at the beginning of this chapter.
- The ALTERNATE_KEY_DEFINITION display shows the record contents in ASCII characters and hexadecimal digits with the alternate-key values underscored. Each alternate key is shown separately by underscores as follows:

- If the concatenated-key or repeating groups attributes are not defined for the key, the underscore characters indicate the alternate-key type (C for collated, I for integer, or U for uncollated).

- If the alternate-key definition specifies repeating groups, the underscores for each alternate-key value in the record include a number (1, 2, and so forth).

- If the key is a concatenated key, the underscores for each key field include one or two letters. The fields concatenated are a_, b_, and so forth up to z_ and then, aa, ba, ca, and so forth.

Examples The following session displays a range of records showing both ASCII and hexadecimal representations.

```
crekf/display_records display_option=both ..
crekf../keys='Everest'..'Kilimanjaro'
                Display_Nested_File      1986-04-23
NOS/VE Keyed File Utilities 1.2 86099     15:08:18
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
Display of records in $MAIN_FILE for:

      COUNT: all
  FIRST_KEY: Everest
   LAST_KEY: Kilimanjaro
Byte: 0          ASCII: E v e r e s t                A s i a
Byte: 0(16)        HEX: 4576657265737420202020202020417369612020202020202020
Byte: 25         ASCII:       8 8 4 8
Byte: 19(16)       HEX: 20202038383438
Byte: 0          ASCII: K 2                          A s i a
Byte: 0(16)        HEX: 4B32202020202020202020202020417369612020202020202020
Byte: 25         ASCII:       8 6 1 1
Byte: 19(16)       HEX: 20202038363131
Byte: 0          ASCII: K i l i m a n j a r o        A f r i c a
Byte: (16)         HEX: 4B696C696D616E6A61726F2020204166726963612020202020
Byte: 25         ASCII:       5 8 9 5
Byte: 19(16)       HEX: 20202035383935
crekf/
```

EXTRACT_RECORDS Subcommand

Purpose Copies records from the currently selected nested file to another file.

Format

EXTRACT_RECORDS or
EXTRACT_RECORD or
EXTR
OUTPUT=file reference or nested-file reference
KEYS=range of key
COUNT=integer
ERROR_LIMIT=integer
STATUS=status_variable

Parameters *OUTPUT or O*

File to which records are copied. You must have at least append permission to the file. If you omit the OUTPUT parameter, records are written to the standard file $OUTPUT.

The OUTPUT parameter for EXTRACT_RECORDS can specify either a sequential or a keyed file. If the output file is a keyed file and a nested file is not specified, the records are copied to the default nested file $MAIN_FILE.

To specify a nested file in a keyed file, enclose the file reference followed by the nested file name in parentheses.

An open position of $BOI indicates that any records existing in the nested file are to be discarded. Do not specify the nested-file name $MAIN_FILE on the OUTPUT parameter when the open_position of the output file is $BOI. (This would request deletion of $MAIN_FILE which is not allowed.)

An open position of $EOI indicates the input records are to be merged into the output file.

KEYS or KEY or K

Optional range of primary-key values to be extracted. The range may be specified in one of the following ways:

- As two primary-key values separated by two periods (..). The first value must be lower than or equal to the second value.

- As one primary-key value specifying the beginning of the range. The number of records extracted is specified by the COUNT parameter.

For an indexed-sequential file, you may use the keywords $FIRST_KEY and $LAST_KEY in a key range to specify the lowest and highest key values, respectively.

Key ranges are not allowed for direct-access files. You can extract records by key value one at a time or use COUNT=ALL.

If you omit the KEYS parameter, no default is assumed and records are extracted beginning with the first record.

COUNT or C

Indicates the number of records to be extracted.

Integer (0 through 4398046511103)	Extracts the specified number of records.
ALL or A	Extracts all records in the nested file.

If you omit the COUNT parameter, the default is 1, unless a range is specified by the KEYS parameter and then it is the number of records in the range. If both a key range and a count are specified, the count limits the number of records extracted in the range.

ERROR_LIMIT or EL

Number of nonfatal (trivial) errors allowed for the EXTRACT_RECORDS operation (integer from 0 through 4398046511103 [2^{42}-1]).

A 0 value indicates no limit; 0 is the default value.

See the Remarks for a description of error processing.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- See the rules listed under Selecting Records at the beginning of this chapter. Also, see the rules for ERROR_LIMIT processing under Adding and Replacing Records From Input Files at the beginning of this chapter.
- Records are extracted only from the currently selected nested file.
- The OUTPUT parameter cannot specify the same file cycle as the CREATE_KEYED_FILE or CHANGE_KEYED_FILE output file, unless the parameter specifies a different nested file in the file cycle.
- EXTRACT_RECORDS supports copying to files with sequential and keyed-file organizations. It does not support copying to byte-addressable files.
- The records are not sorted if the output file is sequential.

  If the output file is a keyed file, records are sorted if certain attributes of the currently selected nested file differ from the EXTRACT_RECORDS output nested file; otherwise, no sorting is performed. The attributes that force sorting are the file_organization, hashing_procedure_name, collate_table_name, key_length, key_type, and key_position.

Examples

This CREATE_KEYED_FILE example extracts records within the key range and then copies the records to NEW_FILE_OUTPUT.

```
/copy_keyed_file $user.new_file
Everest        Asia          8848
K2             Asia          8611
Kilimanjaro    Africa        5895
Matterhorn     Europe        4478
McKinley       North America 6194

/create_keyed_file ..
../output=$user.indexed_sequential_file
crekf/add_records input=$user.new_file
crekf/extract_records output=new_file_output ..
crekf../keys='E'..'Ma', count=3
--INFORMATIVE AA 1275-- The Extract_Records
subcommand of CREATE_KEYED_FILE copied 3 records from
nested file $MAIN_FILE in file
:NVE.USER99.INDEXED_SEQUENTIAL_FILE to
NEW_FILE_OUTPUT.
crekf/copy_file input=new_file_output
Everest        Asia          8848
K2             Asia          8611
Kilimanjaro    Africa        5895
crekf/
```

HELP Subcommand

Purpose Provides online HELP from within the keyed-file utilities.

Format

HELP or
HEL
SUBJECT=string
MANUAL=name
STATUS=status_variable

Parameters

SUBJECT or S

Topic to be located in the online manual index. The topic must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands.

MANUAL or M

Name of the online manual whose index is searched. If you omit the manual parameter, the default is AFM. If the AFM file is not found in the current working catalog, the $SYSTEM.MANUALS catalog is searched.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- If you enter a topic that is not in the manual index, a message appears telling you that the topic could not be found.
- The default manual, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for full-screen applications, the online manual is displayed in screen mode.

  To leave the online manual, use QUIT. To get help on reading the online manual, use HELP.

Examples

This CREATE_KEYED_FILE example accesses the HELP subcommand within the CREATE_KEYED_FILE utility session.

```
crekf/help

The following CREATE_KEYED_FILE subcommands are available:

ADD_RECORDS
REPLACE_RECORDS
COMBINE_RECORDS
EXTRACT_RECORDS
DISPLAY_RECORDS
DELETE_RECORDS
CREATE_NESTED_FILE
SELECT_NESTED_FILE
DELETE_NESTED_FILE
DISPLAY_NESTED_FILE
CREATE_ALTERNATE_INDEXES
QUIT

For a description of a subcommand in the online manual, enter:

HELP subject = '<subcommand>'

To return from an online manual, enter:

QUIT
crekf/
```

QUIT Subcommand

Purpose Terminates the CREATE_KEYED_FILE or CHANGE_KEYED_FILE session and closes the output file.

Format

QUIT or
QUI
STATUS_status_variable

Parameters

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Examples This CHANGE_KEYED_FILE example creates an exact byte-by-byte copy of the input file. The QUIT subcommand terminates the CHANGE_KEYED_FILE utility session.

```
/change_keyed_file ..
../input=$user.new_file ..
../output=$user.indexed_sequential_file
crekf/quit
```

REPLACE_RECORDS Subcommand

Purpose Replaces records in the currently selected nested file.

Format

REPLACE_RECORDS or
REPLACE_RECORD or
REPR
INPUT=file or list of files
SORT=boolean
ERROR_LIMIT=integer
STATUS=status_variable

Parameters

INPUT or I

List of one or more files whose records are to be copied. For each input record, there must be a record in the currently selected nested file with the same primary-key value. You must have at least read access to the input files. This parameter is required.

SORT or S

Indicates whether the input records are sorted before they are added to the currently selected nested file. If you omit the SORT parameter, the default is SORT=TRUE.

ERROR_LIMIT or EL

Number of nonfatal (trivial) errors allowed for the REPLACE_RECORDS operation (integer from 0 through 4398046511103[242-1]).

A 0 value indicates no limit; 0 is the default value.

See the Remarks for a description of error processing.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- Because the REPLACE_RECORDS subcommand does not lock key values, it is recommended that you do not use this subcommand when another task could be updating the same records. If you attach the output file with no write share modes (SHARE_MODES=READ or SHARE_MODES=NONE) before the session, the file cannot be shared for updating.

- See the rules for record copying subcommands listed under Adding and Replacing Records From Input Files at the beginning of this chapter.

Examples This CREATE_KEYED_FILE example replaces records in file $USER.INDEXED_SEQUENTIAL_FILE that have the same primary key.

```
/
Everest       Africa        8800
K2            Asia          8611
Kilimanjaro   Africa        5895

/
Everest       Asia          8848

/
../
crekf/
crekf/
crekf/
                Display_Nested_File                        1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                       13:19:24
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
Display of records in $MAIN_FILE

 Byte: 0            ASCII: Everest       Asia         8848
 Byte: 0            ASCII: K2            Asia         8611
 Byte: 0            ASCII: Kilimanjaro   Africa       5895
crekf/
```

SELECT_NESTED_FILE Subcommand

Purpose Selects the nested file to which subsequent subcomands are to apply.

Format

SELECT_NESTED_FILE or
SELNF
NAME=name
STATUS=status_variable

Parameters

NAME or N

Name of an existing nested file. This parameter is required.

STATUS

Optional SCL status variable. If you specify the STATUS parameter, the command returns its completion status in the specified variable.

Remarks

- To select the default nested file, $MAIN_FILE, specify NAME=$MAIN_FILE.

Examples This CREATE_KEYED_FILE example selects the nested file NESTED_FILE_2 and then displays the list of nested files.

```
crekf/select_nested_file name=nested_file_2
crekf/display_nested_file
                Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                   13:25:31
File = :NVE.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    NESTED_FILE_2            (currently selected nested file)
    NESTED_FILE_1
    $MAIN_FILE
crekf/
```

Keyed-File Recovery 9

Keyed-File Recovery 9

This chapter describes the following topics:

- The means of protecting your keyed files.
- The means of recovering damaged keyed files.
- The Recover_Keyed_File command utility.
- The Administer_Recovery_Log command utility.
- The Restore_Log command utility.

Protecting Your Keyed Files

NOS/VE provides several features that can protect your keyed files against data loss and file structure damage. These features include:

- File and catalog permits to prevent unauthorized access (as described in the NOS/VE System Usage and CYBIL File Management manuals).
- Record locking to coordinate access between instances of open of the same keyed file.
- Attributes that enable recovery of your keyed files in the event of a system failure.

This section describes how to enable file recovery to protect against a system failure. It first describes the types of possible system failure; then how to protect your keyed files against system failures; and finally, how to recover your keyed files when a system failure occurs.

Possible System Failures

System failures that could affect your keyed files can be categorized as processing failures and/or media failures.

A processing failure is the result of an error affecting the CPU or memory, but not mass storage. It can affect your keyed file only if the file is open for updating when the failure occurs. When a processing failure occurs without media failure, the copy of the keyed file on mass storage remains intact, but the following keyed-file errors could result:

- If the failure causes a task abort during an update operation on a keyed file, the operation could be left only partially completed.
- A memory error could lose or corrupt keyed-file blocks that have been modified in memory, but have not yet been copied to mass storage.
- Both of the above failures (partial updates and lost or corrupted file blocks) could result in an incoherent file structure in the keyed file on mass storage.

Unlike a processing failure, a media failure is the result of a data storage device error. It could affect your keyed file by making the file inaccessible, by damaging the file structure, or by corrupting the file data.

All these failures are possible, and different methods are available to protect your keyed files against them. However, each file protection method is optional. If you judge the damage that could result from a type of failure is not worth the cost of protection, you can choose not to use that method of protection. The next section describes the protection methods available.

Protection Methods

The following are the methods of protecting your keyed files against system failures:

- Maintaining frequent backup file copies.
- Enabling the removal of a partial update due to a task abort.
- Forcing the writing of keyed-file blocks from memory to mass storage.

- Maintaining a log of all updates to the keyed file.

More than one method can be used. In fact, the last method, logging of updates, is possible only if backup file copies are maintained.

The following sections describe these protection methods in detail.

Maintaining Backup Copies

In general, sites maintain backup copies of all permanent files. If a system failure causes the loss of a permanent file, the most recent backup copy of the lost file is reloaded. The reload restores the file to its state when the backup copy was written. Any updates to the file since the backup copy was written are lost unless the user has a method of reconstructing them. (One method, using an update log, is described later.)

The more frequently backup copies are written, the fewer updates are lost when a file reload is required. A site usually has a fixed schedule for backups, but you can perform more frequent backups if you wish. You can backup files using the Backup_Permanent_File utility described in the NOS/VE System Usage manual.

Backup_Permanent_File allows you to backup any file you have permission to read but are not required to share in append, modify, or shorten modes. (You cannot backup a file while it could be updated.)

NOTE

It is recommended that you always use the Backup_Permanent_File utility to write backup copies, rather than keeping additional file copies. The RECOVER_FILE_MEDIA subcommand of the Recover_Keyed_File utility can only use backup copies written by the Backup_Permanent_File utility. (The ENABLE_MEDIA_RECOVERY logging option must be enabled before the backup is done.)

Also, a keyed file protected by the ENABLE_MEDIA_RECOVERY option must be backed up after its password, LOG_RESIDENCE attribute, or LOGGING_OPTIONS attribute, is changed.

Enabling Partial Update Removal

An update could be interrupted by a system failure and left partially completed. The update could be a single update operation.

A partially completed update operation could prevent subsequent access to a keyed file if the following scenario occurs:

1. A processing failure occurs while an update operation for a keyed file is in progress.
2. The processing failure causes the task to abort, leaving the update operation only partially completed.
3. The task abort causes an automatic close of the keyed file.
4. The close operation, when it finds the partially completed update, sets the ruined flag for the entire keyed file.
5. Because its ruined flag is set, the keyed file cannot be accessed until it has been re-created.

To protect against this type of failure, you can request that when an automatic close finds a partial update, it removes it, leaving the file as it was before the update operation began.

To request this protection, specify the ENABLE_REQUEST_RECOVERY option in the set of values for the LOGGING_OPTIONS attribute. The LOGGING_OPTIONS attribute is a preserved attribute stored with the file; you can change it using the NOS/VE command CHANGE_FILE_ATTRIBUTES.

To prepare for partial update removal, the system need only keep enough information to remove the last update. Therefore, this failure protection requires little system overhead. The recovery is performed automatically; no additional user processing is required.

Forcing Memory Writes

To understand how forcing memory writes can protect your files, you need to understand how NOS/VE uses memory to modify a mass storage file.

All mass storage files exist as blocks of space allocated on mass storage devices. When a program issues a request to read or modify a file, the block containing the requested data is copied into memory. An update request then modifies the block in memory. By default, only the modified block is copied back to mass storage when the memory space is needed for other data. Therefore, for a short time, the modified file block exists only in memory and not on mass storage.

The process of copying all modified blocks to mass storage is called flushing. A normal close of an open file flushes the file to mass storage. However, if the close discovers that a memory error has caused a modified file block to be lost or corrupted, it sets the altered_not_closed flag for the file. The next attempt to open the file returns the status AAE$ALTERED_NOT_CLOSED, which prevents use of the file because its integrity is in doubt. (The means of recovering from this failure are described in the next section.)

Forced memory writes shorten the time in which modified blocks do not have a mass storage copy. To request forced writing, you change the FORCED_WRITE file attribute value to either TRUE or FORCED_IF_STRUCTURE_CHANGE. The FORCED_WRITE attribute is a preserved attribute stored with the file; you can change its value with the NOS/VE command CHANGE_FILE_ATTRIBUTES.

The default FORCED_WRITE attribute value is FALSE, which requests no forced writing. The two options requesting forced writing are:

FORCED_WRITE=FORCED_IF_STRUCTURE_CHANGE

This option requests that each time a request changes more than one file block (that is, it changes the file structure), the modified blocks are copied to mass storage at completion of the request.

With this option, in most cases, a memory failure would leave the file structure intact. All single-block updates since the last forced-write could be lost. Also, if a file structure change is interrupted by the system failure, the file structure could be damaged, but the existing file records would still be in order and, therefore, could be salvaged.

FORCED_WRITE=TRUE

This option requests that each time a request changes a file block, the modified block is copied to mass storage at completion of the request.

With this option, a memory failure could lose only the update in progress for each instance of open of the file. However, if the lost update is a file structure change, the file structure could be damaged and its restoration required.

For Better Performance

Forced memory writes add to the I/O time for the keyed file. In particular, the FORCED_WRITE=TRUE option causes a small increase for random updates, but a severe increase for sequential updates.

Maintaining Update Logs

When a file is damaged, the best recovery available may be to replace the file with a backup copy. However, complete restoration of the file would require a means of reconstructing all updates to the file since the backup copy was written.

The most reliable method for reconstructing file updates is through use of an update log that contains a record for each update to the file. By reading the log records in chronological order, the updates can be performed on the backup copy in the same order they were performed on the original file. This results in a complete restoration of the damaged file.

To request maintenance of an update log for a keyed file, you request the option ENABLE_MEDIA_RECOVERY in the set of values for the LOGGING_OPTIONS file attribute. By default, the update log for the file is kept on catalog $SYSTEM.AAM.SHARED_RECOVERY_LOG. However, you can specify the catalog to contain the log as the LOG_RESIDENCE file attribute. (The log must first be created by the Administer_Recovery_Log utility as described later in this chapter.)

The LOGGING_OPTIONS and LOG_RESIDENCE attributes are preserved attributes stored with the file; you can change their values using the NOS/VE command CHANGE_FILE_ATTRIBUTES.

NOTE

It is not recommended that the default log, $SYSTEM.AAM.SHARED_RECOVERY_LOG, be used extensively for logging update operations. In general, you should specify a different LOG_RESIDENCE for vital applications. This enables you to isolate the effects of a media failure on the log.

Also, after changing the file password, the LOG_RESIDENCE attribute, or the LOGGING_OPTIONS attribute, you must request a backup of the keyed file before the file is updated again.

The Backup_Permanent_File utility records the backup as an entry on the file log as well as on the default log, $SYSTEM.AAM.SHARED_RECOVERY_LOG. The entry for the backup copy marks the point in the log at which reading of the log should begin when the keyed file is reconstructed using that backup copy. The log entry states when the backup was performed and where the backup file resides.

You should also maintain backup copies of the log itself because it, too, could be damaged by a system failure. (A damaged log can be restored using the Restore_Log utility.) The log is backed up automatically as specified by the log attributes. (The log attributes are specified when the log is created by the Administer_Recovery_Log utility.)

Recovering Your Keyed Files

This section describes strategies for determining whether your keyed files are damaged and, if they are damaged, methods of file recovery. The recovery method used depends on the type of file damage and whether a usable backup copy and log are available for the file.

Recovering From a Processing Failure

Usually, you are notified of a processing failure when an application terminates abnormally and one or more messages indicate that the task aborted due to a system failure. If the application was using one or more files when the failure occurred, you should determine if the files were damaged and if file updates were lost. To do so:

1. Attempt to attach the file. If a permanent file cannot be found, the mass storage copy of the file has been lost, indicating a media failure. (See the discussion under Recovering from a File Media Failure.)

2. Attempt to open and read the file. (For example, you could attempt to copy it.) If the attempt fails with a message stating that the file is ruined or unusable, a flag has been set indicating that the file structure may be damaged.

   If a usable backup copy and update recovery log are available for the file, attempt to recover the file using Recover_Keyed_File. If not, you could attempt to clear the error flag by duplicating the file using a COPY_KEYED_FILE command. (The Recover_Keyed_File utility is described later in this chapter. The COPY_KEYED_FILE command is described in chapter 6.)

3. If you use COPY_KEYED_FILE to clear the error flag, you should determine if any updates were lost. For example, if you know what the last update should have been, you should read that record to determine if it was updated. If updates are missing, re-run the updates.

Recovering From a File Media Failure

A media failure can damage your files even if the file is not in use. You may become aware of a media failure when:

- Site personnel notify you that your file was lost and its most recent backup copy has been loaded in its place.

- You cannot attach your file because the permanent file manager cannot find it.
- Even though no processing failure has occurred, an attempt to open your file returns a message indicating that the file is ruined.
- An application finds that your file contains bad data.

If the file has not been updated since the last backup copy was written, a reload of the backup copy restores the file.

If the file has been updated since the last backup copy, determine if a usable log is available for the file. If a log is available, reconstruct the updates using the log. To do so, execute the Recover_Keyed_File utility with the RECOVER_FILE_MEDIA subcommand.

NOTE

Once a keyed file is recovered using the Recover_Keyed_File utility with the RECOVER_FILE_MEDIA subcommand, it must be backed up (using the Backup_Permanent_File utility) before it can be updated.

If the file has been updated since the last backup copy, but no log is available, the file might be usable again if the the file structure error flag can be cleared. To do so, duplicate the file using the COPY_KEYED_FILE command described in chapter 6.

Also, if no log is available, any lost updates must be restored by other means. If the damaged file is available, attempt to salvage the data records from it.

Recover_Keyed_File Utility

The Recover_Keyed_File utility is an NOS/VE command utility with which you can initiate recovery attempts for a damaged keyed file.

A Recover_Keyed_File utility session begins with a RECOVER_KEYED_FILE command followed by a subcommand for the recovery attempt. The session ends when a recovery attempt returns a fatal error or when you enter the QUIT subcommand to end the session.

Currently, the Recover_Keyed_File subcommands are as follows:

RECOVER_FILE_MEDIA

Reloads a backup of the file and then updates it using an update recovery log for the file.

VOID_LOG_FOR_RESTORED_FILE

Discards the update recovery log associated with a restored file.

HELP

Provides access to online information about the utility.

QUIT

Ends the session.

RECOVER_KEYED_FILE Command

Purpose Begins a keyed-file recovery attempt.

Format

RECOVER_KEYED_FILE or
RECKF
FILE=file
PASSWORD=name or NONE
STATUS=status_variable

Parameters

FILE or F

File path to the damaged keyed file to be recovered. This parameter is required.

If the damaged file does not currently exist, its cycle number cannot be determined by default. Therefore, the file path must explicitly specify the file cycle number so that the utility can reload the correct backup copy.

PASSWORD or PW

File password specified when Backup_Permanent_File wrote the backup copy of the file. A file password is optional, but, if a password exists for the file, it is required on this command. If no password exists for the file, NONE can be specified.

NOTE

The file password in effect when the backup copy was written must be the same password in effect when the file was damaged. Otherwise, the backup copy cannot replace the damaged file.

STATUS

Optional SCL status variable in which the completion status of the command is returned. Its use depends on whether the command is executed in a batch job or an interactive session.

In a batch job, when an error is returned for a subcommand without its own status variable, the error is stored in the Recover_Keyed_File status variable, if any, and the session terminates.

In an interactive session, only the completion status of the Recover_Keyed_File command and its QUIT subcommand are stored in the RECOVER_KEYED_FILE status variable. Errors returned by subcommands are displayed at the terminal and the session continues.

Remarks

- The LOG_RESIDENCE attribute of the file specified on the command must match the LOG_RESIDENCE attribute of the backup copy to be reloaded. Recover_Keyed_File cannot use a backup copy that was written before the LOG_RESIDENCE attribute of the file was changed.

- If the file does not currently exist and the LOG_RESIDENCE of its backup copy is not the default log, you must enter a SET_FILE_ATTRIBUTE command for the file. The command must specify the same file cycle specified on the RECOVER_KEYED_FILE command and the same LOG_RESIDENCE as that of the backup copy to be used. (See the Example.)

- Similarly, if the file does not currently exist, but the file had a password when the backup copy was written, you must create the file with the same password. To do so, enter a CREATE_FILE command specifying the file path (including its cycle number) and the PASSWORD parameter. (See the Example.)

Examples

The following session attempts to restore a keyed file that no longer exists using its latest backup copy. When the latest backup copy was written, the file password was HUSH_HUSH and the LOG_RESIDENCE attribute was $USER.MY_LOG. Therefore, those values must be reestablished for the file cycle.

```
/recover_keyed_file, $user.keyed_file.1
reckf/create_file, $user.keyed_file.1, ..
reckf../password=hush_hush
reckf/set_file_attribute, $user.keyed_file.1, ..
reckf../log_residence=$user.my_log
reckf/recover_file_media

--INFORMATIVE AA 1495-- Log
$SYSTEM.AAM.AAF$SHARED_RECOVERY_LOG:
Recover_File_Media is attempting to find a backup
of file :NVE.KEYED_FILE.1 in this log.
```

```
--WARNING AA 1410-- IMPORTANT - File :NVE.MY_
KEYED_FILE.1 : RECOVER_KEYED_FILE is deleting this
file to ensure there is disk space for the backup
file to be reloaded.  To restart RECOVER_KEYED_
FILE, if it terminates before the file is
successfully restored, you must supply the full
file path, including the cycle number (1).

--WARNING AA 1480-- IMPORTANT - File :NVE.MY_
KEYED_FILE.1 : The cycle number of this file
is 1.

--WARNING AA 1415-- IMPORTANT - The LOG_RESIDENCE
should also be set to :NVE.MY_LOG using the
SET_FILE_ATTRIBUTES command if RECOVER_KEYED_FILE
is restarted for this file.

--INFORMATIVE AA 1380-- Recover_Keyed_File is now
attempting to restore :NVE.KEYED_FILE.1 from the
backup file.

--INFORMATIVE AA 1495-- Log :NVE.MY_LOG :
Recover_File_Media is attempting to find a backup
of file :NVE.MY_KEYED_FILE.1 in this log.

--INFORMATIVE AA 1500-- Recover_File_Media has
successfully reloaded file :NVE.KEYED_FILE.1
using the backup record from log :NVE.MY_LOG.
The date of the backup was February 10, 1987
at 2:57 PM.

--INFORMATIVE AA 1385-- Recover_Keyed_File is now
applying the changes from the log to the file
restored from the backup.

--INFORMATIVE AA 1465-- File :NVE.KEYED_FILE.1 :
Recover_File_Media processed 4 records from the log.
There were 0 trivial errors.

reckf/quit
/
```

RECOVER_FILE_MEDIA Subcommand

Purpose Reloads a backup of the file and then updates it using an update recovery log for the file.

Format

RECOVER_FILE_MEDIA or
RECFM
DAYS_SINCE_LAST_GOOD=integer
HOURS_SINCE_LAST_GOOD=integer
MINUTES_SINCE_LAST_GOOD or MSLG=integer
FILE_CLASS=character
INITIAL_VOLUME=name
STATUS=status variable

Parameters

DAYS_SINCE_LAST_GOOD or DSLG

Number of days since the damaged file was intact (any integer not less than 0). It is used with the next two parameters to determine the backup copy to be reloaded.

If the first three parameters are omitted, the default value for each is 0, causing the latest backup copy to be reloaded.

HOURS_SINCE_LAST_GOOD or HSLG

Number of hours (added to the days specified by the first parameter) since the damaged file was intact (an integer from 0 through 23).

If the first three parameters are omitted, the latest backup copy is reloaded.

MINUTES_SINCE_LAST_GOOD or MSLG

Number of minutes (added to the days and hours specified by the first two parameters) since the damaged file was intact (an integer from 0 through 59).

If the first three parameters are omitted, the latest backup copy is reloaded.

FILE_CLASS or FC

Specifies the class of the file to be assigned. Refer to the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual, Volume 2, for class assignments and a complete description of this parameter.

INITIAL_VOLUME or IV

Name specifying the volume serial number (VSN) of the mass storage volume to which the file is to be assigned. The name is specified as a string of from 1 through 6 characters. Refer to the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual, Volume 2, for a complete description of this parameter.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- This subcommand is effective only if both a backup copy and an update recovery log are available for the file.
- An update recovery log is maintained for the file only if its LOGGING_OPTIONS attribute includes the option ENABLE_MEDIA_RECOVERY.
- Only backup copies created by the Backup_Permanent_File utility can be reloaded because those backup copies are recorded in the update recovery log for the file. (The ENABLE_MEDIA_RECOVERY logging option must be set before the backup.)
- For a backup copy to be used, the file password (if any), the LOG_RESIDENCE attribute, and the LOGGING_OPTIONS attribute for the file must not have changed since the backup copy was written.
- The FILE_CLASS and INITIAL_VOLUME parameters are described in detail as parameters of the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual, Volume 2.

NOTE

Before entering the RECOVER_FILE_MEDIA subcommand, write down the following information:

- The file cycle number.
- The file password, if any.
- The LOG_RESIDENCE attribute of the file if it is not the default log.

  If the recovery attempt fails (because of a system crash), the file may have been deleted. Because the file no longer exists, if reattempting the reload you must provide Recover_Keyed_File with the above information, as follows:

- Specify the file cycle number on the file path.
- Specify the file password on a CREATE_FILE command for the file cycle.
- Specify the LOG_RESIDENCE attribute on a SET_FILE_ATTRIBUTE command.

- The recovery attempt proceeds as follows:

  1. The specified file is checked for a keyed file whose LOGGING_OPTIONS attribute includes ENABLE_MEDIA_RECOVERY.
  2. The system log is searched for the entry of the most recent backup of the file cycle before the time specified on the subcommand.

     If the backup copy is on magnetic tape, it requests the appropriate tape volume. The reloaded file replaces the existing file copy.

     If the LOG_RESIDENCE attribute indicates a user log (rather than the default system log), the same search is performed on the user log as the search described for the system log in step 2, looking for an entry of a more recent usable backup copy.

The Restore_Permanent_File Utility is called to reload the most recent backup copy indicated by the system and/or user log. Validation of the restored backup file is performed to ensure that the correct file has been restored.

3. The updates recorded in the log are applied to the backup copy in the order recorded on the log, starting from the time of the restored backup.

   Only updates recorded on the log are performed.

- Progress messages are issued as recovery proceeds. Be sure to read the messages as they appear.
- Once a keyed file is recovered using RECOVER_FILE_MEDIA, it must be backed up (using the Backup_Permanent_File utility) before it can be updated.

Examples The following session recovers the file using the last backup copy.

```
/recover_keyed_file, $user.my_keyed_file
reckf/recover_file_media

--INFORMATIVE AA 1495-- Log $SYSTEM.AAM.AAF$SHARED_
RECOVERY_LOG : Recover_File_Media is attempting to
find a backup of file :NVE.MY_KEYED_FILE.1 in this
log.

--WARNING AA 1410-- IMPORTANT - File :NVE.MY_KEYED_
FILE.1 : RECOVER_KEYED_FILE is deleting this file to
ensure there is disk space for the backup file to be
reloaded.  To restart RECOVER_KEYED_FILE, if it
terminates before the file is successfully restored,
you must supply the full file path, including the
cycle number (1).

--WARNING AA 1480-- IMPORTANT - File :NVE.MY_
KEYED_FILE.1 : The cycle number of this file is 1.

--WARNING AA 1415-- IMPORTANT - The LOG_RESIDENCE
should also be set to :NVE.MY_LOG using the
SET_FILE_ATTRIBUTES command if RECOVER_KEYED_FILE
is restarted for this file.

--INFORMATIVE AA 1380-- Recover_Keyed_File is now
attempting to restore :NVE.MY_KEYED_FILE.1 from
the backup file.

--INFORMATIVE AA 1495-- Log :NVE.MY_LOG :
Recovery_File_Media is attempting to find a
backup of file :NVE.MY_KEYED_FILE.1 in this
log.

--INFORMATIVE AA 1500-- Recover_File_Media has
successfully reloaded file :NVE.MY_KEYED_FILE.1
using the backup record from log :NVE.MY_LOG.
The date of the backup was February 10, 1987
at 2:57 PM.

--INFORMATIVE AA 1385-- Recover_Keyed_File is now
applying the changes from the log to the file
restored from the backup.
```

```
--INFORMATIVE AA 1465-- File :NVE.MY_KEYED_FILE :
Recover_File_Media processed 4 records from the log.
There were 0 trivial errors.

reckf/quit
/
```

VOID_LOG_FOR_RESTORED_FILE Subcommand

Purpose Discards the update recovery log associated with a file that has been restored using the Restore_Permanent_File utility.

Format **VOID_LOG_FOR_RESTORED_FILE** or **VOILFRF**

STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- This subcommand is provided for situations in which an older version of the file is restored using the Restore_Permanent_File utility, and the user, content with this version, does not want to try to recover lost updates from the log.
- Updates cannot be recorded on a log that is associated with a restored file. This is because the updates on the log do not correspond to the restored version (usually an older version) of the file. Therefore, this subcommand is used to discard all past logged updates for the restored file.
- After the update recovery log is discarded, a backup copy of the file must be created by the Backup_Permanent_File utility if subsequent updates are to be recorded on the log.
- An update recovery log is maintained for a file only if its LOGGING_OPTIONS attribute includes the option ENABLE_MEDIA_RECOVERY.

Examples The following session discards the update recovery log associated with the restored file $USER.KEYED_FILE:

```
/recover_keyed_file, $user.keyed_file
reckf/void_log_for_restored_file

--File : V02.SONYA.KEYED_FILE : Void_Log_For_
Restored_File has successfully discarded all
previous log information for this file.
Logging will resume after you have performed
a BACPF of this file.

reckf/quit
/
```

HELP Subcommand

Purpose Provides access to online information about the utility.

Format

HEL
SUBJECT=string
MANUAL=file
STATUS=status variable

Parameters

SUBJECT or S

Topic to be found in the index of the online manual. The topic must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands and prompts for display of a subcommand description in the online manual.

MANUAL or M

Online manual file to be read. If you omit the MANUAL parameter, the default is AFM. The working catalog is searched for the AFM file and then the $SYSTEM.MANUALS catalog.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

If STATUS is omitted, the completion status is returned to the terminal in an interactive session or, in a batch job, to the status variable specified on the RECOVER_KEYED_FILE command, if any.

Remarks

- If the SUBJECT parameter specifies a topic that is not in the manual index, a nonfatal error is returned notifying you that the topic could not be found.
- The default manual file, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for full-screen applications, the online manual is displayed in screen mode. Help is available for reading the online manual. To leave the online manual and return to the utility, use QUIT.

Examples The following session shows the default display returned by the HELP subcommand.

```
/recover_keyed_file, $user.keyed_file.1
reckf/help

The following Recover_Keyed_File subcommands are
available:

RECOVER_FILE_MEDIA
VOID_LOG_FOR_RESTORED_FILE
HELP
QUIT

For a description of a subcommand in the online
manual, enter:

HELP subject = '<subcommand>'

To return from an online manual, enter:

QUIT
reckf/quit
/
```

QUIT Subcommand

Purpose Ends the Recover_Keyed_File session.

Format

QUIT or
QUI
STATUS=status variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the command is returned.

If STATUS is omitted, the completion status is returned to the terminal in an interactive session or, in a batch job, to the status variable specified on the RECOVER_KEYED_FILE command, if any.

Remarks

- The QUIT command is required to end a session.
- A recovery attempt that returns a fatal error ends the session.

Examples The following session is ended after a recovery attempt fails.

```
/recover_keyed_file, $user.my_keyed_file
reckf/recover_file_media

--INFORMATIVE AA 1495-- Log $SYSTEM.AAM$SHARED_
RECOVERY_LOG : Recover_File_Media is attempting
to find a backup of file : V01.KF4077.MY_KEYED_
FILE.1 in this log.

--WARNING AA 1410-- IMPORTANT - File: :
V01.KF4077.MY_KEYED_FILE.1 :
RECOVER_KEYED_FILE is deleting this file to ensure
there is disk space for the backup file to be
reloaded.  To restart RECOVER_KEYED_FILE, if it
terminates before the file is successfully
restored, you must supply the full file path,
including the cycle number (1).

--WARNING AA 1480-- IMPORTANT - File
:V01.KF4077.MY_KEYED_FILE.1 : The cycle
number of this file is 1.
```

```
--WARNING AA 1415-- IMPORTANT - The LOG_RESIDENCE
should also be set to :V01.KF4077.MY_LOG using the
SET_FILE_ATTRIBUTES command if RECOVER_KEYED_FILE
is restarted for this file.

--INFORMATIVE AA 1380-- RECOVER_KEYED_FILE is now
attempting to restore : V01.KF4077.MY_KEYED_
FILE.1 from the backup file.

--INFORMATIVE AA 1495-- Log : V01.KF4077.MY_LOG :
Recover_File_Media is attempting to find a backup
of file :V01.KF4077.MY_KEYED_FILE.1 in this log.

--Log $SYSTEM.AAM.SHARED_RECOVERY_LOG :
No record was found in this log for a backup of
file :V01.KF4077.MY_KEYED_FILE.1 prior to
July 7, 1987 at 3:53 PM.

--FATAL-- File :V01.KF4077.MY_KEYED_FILE.1
Could not be recovered because the log does
not contain a log record for a valid backup
of this file.

reckf/quit
/
```

Administer_Recovery_Log Utility

The Administer_Recovery_Log utility is the NOS/VE command utility used to create and maintain the logs used by the keyed-file interface.

An update recovery log is the log on which all update operations for a file are recorded when the LOGGING_OPTIONS attribute of the file includes ENABLE_MEDIA_RECOVERY.

Any number of keyed files can use the same log; the log entries contain a unique identifier [a signature] for the file to which they apply.

A log cannot be used until it has been created by the Administer_Recovery_Log utility.

Ring Constraints

The Administer_Recovery_Log utility enforces the following ring constraints:

- To modify a log, the Administer_Recovery_Log utility must be executed in a ring at least as privileged as the ring in which the log was created.
- To create or modify the default system log, you must execute the Administer_Recovery_Log utility in ring 6 or lower (at the system console). The default system log, $SYSTEM.AAM.SHARED_RECOVERY_LOG, is created during system installation.

Although the Administer_Recovery_Log utility enforces the ring constraints described above, the files the utility creates have the ring attributes (4,4,4). Thus, files created using the Administer_Recovery_Log utility can only be deleted using the Administer_Recovery_Log utility.

Administer_Recovery_Log Tasks

The Administer_Recovery_Log utility can perform the following tasks:

Task	Applicable Subcommands
Delete existing logs	DELETE_LOG QUIT
Create a new log, or change an existing log	USE_LOG CONFIGURE_LOG_RESIDENCE SET_LOG_BACKUP_ACCOUNT CONFIGURE_LOG_BACKUP SET_PERFORMANCE_OPTION SET_VERIFICATION_LEVEL DISPLAY_LOG_CONFIGURATION CANCEL_LOG_CHANGES QUIT
Display and clear the problem journal	USE_LOG DISPLAY_PROBLEM_JOURNAL CLEAR_PROBLEM_JOURNAL QUIT
Backup a log immediately	USE_LOG BACKUP_LOG QUIT

Creating a New Log

The following steps outline the process of creating a new log.

1. In general, the ADMINISTER_RECOVERY_LOG command creates the catalog specified on the command if it does not exist. However, if you are creating a log in a catalog that belongs to another user, the other user must create the catalog and give you access to it. (The commands to do so are described in the NOS/VE System Usage manual.) For example, assuming the user name of the log administrator is LOG_ADMINISTRATOR, the following commands create the catalog $USER.MY_LOG:

```
create_catalog, $user.my_log
create_catalog_permit, $user.my_log, group=user, ..
..user=log_administrator, access_modes=(all, cycle, control)
```

2. Begin the Administer_Recovery_Log session by entering an ADMINISTER_RECOVERY_LOG command. For example:

```
administer_recovery_log
```

3. Specify the catalog in which the new log is to be created using the USE_LOG subcommand. Administer_Recovery_Log creates a new log only if the specified catalog does not already contain a log. (If it already contains a log, the session modifies the existing log [see Modifying an Existing Log later in this chapter].) For example:

```
use_log, $user.my_log
```

4. Establish the configuration of the log using the CONFIGURE_LOG_RESIDENCE subcommand. For example, to configure the log to use four (4) repositories, perform the following:

```
configure_log_residence, repositories=4
```

5. To have the log backed up automatically, perform this step:

   a. Specify the account to be used by the jobs the system initiates to backup the log. This is done using the SET_LOG_BACKUP_ACCOUNT subcommand. For example:

```
set_log_backup_account, user=sonya, password=uquiwi
```

   b. Specify a backup file for each repository specified on the CONFIGURE_LOG_RESIDENCE subcommand using CONFIGURE_LOG_BACKUP subcommands. For example, if you specified that the log use four (4) repositories, you must enter four CONFIGURE_LOG_BACKUP subcommands.

NOTE

If you use the CONFIGURE_LOG_BACKUP subcommand to backup files to tape (which is the default), the tape is mounted and validated immediately, and the file is backed up at that time. Also, the tape used must be labeled. Labeling of tapes is done at the system console by the system operator.

```
configure_log_backup, backup1, recorded_vsn='tp1'
conlb, backup2, rvsn='tp2'
conlb, backup3, rvsn='tp3'
conlb, backup4, rvsn='tp4'
```

6. If desired, you can set performance and verification options for the log using the SET_PERFORMANCE_OPTION and SET_VERIFICATION_LEVEL subcommands. (The SET_VERIFICATION_LEVEL subcommand is effective only when creating a new log; it cannot be changed after the log is created.) For example:

```
set_performance_option, emphasis=reliability
set_verification_level, verify_log_entries=yes
```

7. Check that the log specifications are as desired by entering the DISPLAY_LOG_CONFIGURATION subcommand. For example:

```
admrl/display_log_configuration

Administer_Recover_Log                          1987-06-05
NOS/VE Keyed File Utilities 1.4 87156             15:37:37

Log_Control_File = :NVE.SONYA.MY_LOG.AAF$LOG_CONTROL_FILE.1

Number_Of_Repositories        : 4
Current_Repository            : 0
Repository_Expiration_Time    :
Repository_Size_Limit         : 100000000 "bytes"
Repository_Switching_Size     : 70000000 "bytes"
Repository_Switching_Time     : 1440 "minutes"
Switch_Suppression_Size       : 0 "bytes"
Switch_Suppression_Time       : 0 "minutes"
Oldest_Valid_Entry            :

Performance_Option
  Parcel                      : Reliability
  Record                      : Reliability
Verify_Log_Entries            : Yes

Backup_Account
  User                        : SONYA
  Family_Name                 : UQUIWI
  User_Job_Name               :
  Job_Class                   :
  Account                     :
  Project                     :
  Output_Disposition          : PRINTER
  User_Information            :
```

```
Log_Backup_Pool
  Tape_File_Name                : BACKUP1
                               [ next available log backup tape ]
    Media                       : Tape
    Device_Type                 : MT9$6250
    Count_of_VSNs               : 1
      External_VSN              : tp1
      Recorded_VSN              : tp1

  Tape_File_Name                : BACKUP2
    Media                       : Tape
    Device_Type                 : MT9$6250
    Count_of_VSNs               : 1
      External_VSN              : tp2
      Recorded_VSN              : tp2

  Tape_File_Name                : BACKUP3
    Media                       : Tape
    Device_Type                 : MT9$6250
    Count_of_VSNs               : 1
      External_VSN              : tp3
      Recorded_VSN              : tp3

  Tape_File_Name                : BACKUP4
    Media                       : Tape
    Device_Type                 : MT9$6250
    Count_of_VSNs               : 1
      External_VSN              : tp4
      Recorded_VSN              : tp4

      Damage_Conditions         : None
```

8. To change a specification, re-enter the appropriate subcommand with the correction. Or, you can cancel all specifications and delete requests for the log with the CANCEL_LOG_CHANGES subcommand or by specifying APPLY_LOG_CHANGES=FALSE on the QUIT subcommand.

   With the exception of the performance option (SET_PERFORMANCE_OPTION) and backup account information (SET_LOG_BACKUP_ACCOUNT), the configuration of the log cannot be changed after it is created by Administer_Recovery_Log.

9. If the log specifications are correct, enter the QUIT subcommand to apply the specifications and end the session. For example:

```
quit
```

10. You should immediately backup the new log specifications. To do so, use the Backup_Permanent_File utility to backup the file AAF$LOG_CONTROL_FILE in the log catalog. For example, the following session uses the backup file $USER.LOG_CONTROL_BACKUP to backup the log specifications for $USER.MY_LOG:

```
/backup_permanent_file, $user.log_control_backup, ..
../list=$output
PUB/backup_file, $user.my_log.aaf$log_control_file
PUB/quit
/
```

    The backup file written by these commands is reloaded only if a system failure damages the log before it is put into use and backed up by an automatic log backup job.

11. The owner of the catalog containing the new log must give the following permissions to all users who can update keyed files that use the log.

```
Access_Modes=(read, write)
Share_Modes=none
```

    For example, the following command allows all users to use the log in $USER.MY_LOG:

```
create_catalog_permit, $user.my_log, group=public, ..
  access_modes=(read, write), share_modes=none
```

Modifying an Existing Log

This section describes how to modify an existing log.

To modify an existing log:

1. Begin the session by entering an ADMINISTER_RECOVERY_LOG command.

2. Specify the log to be changed using the USE_LOG subcommand.

3. Display the existing log specifications with the DISPLAY_LOG_CONFIGURATION subcommand.

4. Enter the changes using one or more of the following subcommands:

```
SET_LOG_BACKUP_ACCOUNT
SET_PERFORMANCE_OPTION
```

5. Display the new log specifications with the DISPLAY_LOG_CONFIGURATION subcommand.

6. To change a specification, re-enter the appropriate command. Or, you can cancel all changes and delete requests for the log with the CANCEL_LOG_CHANGES subcommand or by specifying APPLY_LOG_CHANGES=FALSE on the QUIT subcommand.

7. If the log changes are correct, enter the QUIT subcommand to carry out the changes and end the session.

8. You should immediately backup the log changes. To do so, use the Backup_Permanent_File utility to backup the file AAF$LOG_CONTROL_FILE in the log catalog.

Configuring a Log

As stated before, a log is a catalog of files. One log, identified by its catalog path, can be used by any number of keyed files.

The NOS/VE log manager software identifies the entries made to the log such that the collection of entries written as the update recovery log for a keyed file can be considered logically separate from all other log entries. Thus, that collection of entries can be referred to as a logical log. All entries in a logical log have the same unique identifier (called its signature).

When the Administer_Recovery_Log utility configures a new log or changes an existing log configuration, it is configuring the physical log, not a particular logical log. Therefore, the configuration affects all keyed files which refer to that log in their LOG_RESIDENCE attribute.

Log Repositories

The log configuration describes the repositories in the log and provides for backups of those repositories. A repository is a log file in which log entries are written. Each log must have two or more repositories. If the log is configured for automatic backups, at least three repositories are required.

Only one repository is active at a time. Data is written to the active repository until the system determines that a repository switch should occur.

Repositories are switched in round-robin order. In other words, each repository is written in order until all have been written and then each repository, starting with the first, is overwritten in order. So, the number of entries stored in a log at any time is the sum of the entries in its repositories.

Repository Size Limits

The overall log length is the sum of the lengths of its repositories. Therefore, the number of repositories and the limits on the size of the repositories determine the maximum length of the log.

NOTE

A log must be long enough to record all updates to the keyed files in the interval between backups of the keyed files. For example, if the keyed files using the log are backed up daily, the log must be able to record at least a full day of updates to those files. A substantial margin is recommended.

Also, you must consider a recovery situation in which users are recovering updates from the oldest repositories while other users are simultaneously recording log entries. In this situation, a repository switch is made to the oldest repository for purposes of recording log entries while the same repository is being read for recovery purposes. The log entries continue at the expense of the recovery. Configuring the log with extra repositories (two extra repositories are usually enough) is therefore recommended to avoid this conflict.

The length of a repository is the number of bytes of data written to it before the system switches to the next repository. Therefore, the repository length is determined by the parameters that determine when the repository is switched. (The parameters are specified by the CONFIGURE_LOG_RESIDENCE subcommand.)

An automatic repository switch occurs when either of the following events occurs:

- When the repository size reaches the REPOSITORY_SWITCHING_SIZE for the log, the system checks whether the SWITCH_SUPPRESSION_TIME has passed since the last repository switch. If so, a switch occurs.

- When the time since a repository switch reaches the REPOSITORY_SWITCHING_TIME for the log, the system checks whether the SWITCH_SUPPRESSION_SIZE has been reached. If so, a switch occurs.

The REPOSITORY_SIZE_LIMIT parameter sets an absolute limit on the size of any repository in the log. During normal log activity, the active repository size should never approach the REPOSITORY_SIZE_LIMIT.

Estimating Repository Size

When estimating an appropriate repository size, you should assume that each log entry is approximately 150 bytes plus the record data and the nonembedded primary-key value, if any. You also need to know the number of keyed files that will be using the log and the expected update rates (number of updates per hour) for each file. This allows you to estimate the repository size required to hold the log entries for a given period of time.

For example, suppose only 5 keyed files will use the log and the number of updates per hour is assumed to be no more than 100 per file. If a repository should hold the log entries for approximately 4 hours, it should hold approximately 2000 log entries. Assuming that, for each file, the record length is 80 bytes and the primary key is embedded, the expected log entry size would be 230 bytes (150 + 80). Therefore, the length of a repository for 2000 log entries would be approximately 460,000 bytes.

Log Backup Files

To protect the log from system failures, you can configure backup files for the log. If backup files are configured, each repository switch initiates a backup job to backup the repository just switched from.

For example, suppose the log has four repositories and a repository switch from repository 2 to repository 3 occurs. If backup files are configured for the log, a backup job is started. It copies repository 2 to a backup file.

The number of backup files, if any, must be the same as the number of repositories for the log. Only one backup file is used by each backup job. The backup files are used in round-robin order. After all backup files are used, the next backup job overwrites the oldest backup file.

For example, suppose a log has four repositories (1, 2, 3, and 4) and four backup files (A, B, C, and D). The repositories on each backup file would be as follows:

Backup File	Switch	Used for Repositories
A	1 to 2	1
B	2 to 3	2
C	3 to 4	3
D	4 to 1	4

Each backup file must be at least as long as one repository plus the log control file.

The log backup configuration specifies the backup files and whether each backup file is a mass storage file or a magnetic tape file:

- If the file is a magnetic tape file, the configuration specifies the tape volumes that comprise the file and the tape density. These parameters determine the amount of data that can be recorded on the tape file.
- If the file is a mass storage file, the maximum length of the file is the maximum file length allowed the user to which the file belongs.

Log_Temporarily_Full Status

Typically a switch from one repository to the next occurs when REPOSITORY_SWITCHING_SIZE or REPOSITORY_SWITCHING_TIME is reached. However, if SWITCH_SUPPRESSION_TIME is set to other than its default value of 0 (no suppression), switching could be suppressed such that the active repository grows beyond REPOSITORY_SWITCHING_SIZE. Such growth cannot proceed forever; if it reaches the midpoint between REPOSITORY_SWITCHING_SIZE and the REPOSITORY_SIZE_LIMIT, no more update operations will be allowed for the files using the log.

Each update of a keyed file returns the nonfatal status AAE$LOG_TEMPORARILY_FULL. This continues until a repository switch occurs.

When users are receiving the AAE$LOG_TEMPORARILY_FULL status, the log administrator can clear the status using the BACKUP_LOG subcommand of the Administer_Recovery_Log utility. The BACKUP_LOG subcommand initiates an immediate repository switch, as part of the backup process.

However, the log administrator should also determine why the AAE$LOG_TEMPORARILY_FULL status occurred. It may indicate that:

- An unusually large number of updates were being made in a relatively short time to the files that use the log, or
- The log is too small for the normal update activity of the keyed files that use it, or
- SWITCH_SUPPRESSION_TIME is too large for the normal update activity of the keyed files that use it.

The following configuration changes would decrease the frequency of the AAE$LOG_TEMPORARILY_FULL status. (The following changes can be made only to new logs. If the log already exists, it must be deleted, then recreated.)

- Decrease the SWITCH_SUPPRESSION_TIME interval so that a switch can occur sooner.

  Care should be taken not to reduce the lifespan of the log (the length of time covered by the log entries in the log) to less than the time between backups of the keyed files using the log.

- Increase REPOSITORY_SWITCHING_SIZE and/or REPOSITORY_SIZE_LIMIT so that more logging can be done on a repository (making it more likely that SWITCH_SUPPRESSION_TIME will expire before the AAE$LOG_TEMPORARILY_FULL condition occurs).

  This will cause more log entries to reside on each repository, which has certain drawbacks. For one, more disk space is consumed by the log. For another, if the log is configured for backups, at any given time there will be more log entries (a repository's worth) that are not on a backup (furthermore, the individual backups will be larger).

Besides the changes described above, the log administrator may choose to increase the overall log size by increasing the number of repositories. This would require additional backup files (if backup files are included in the configuration).

ADMINISTER_RECOVERY_LOG Command

Purpose Begins an Administer_Recovery_Log utility session.

Format

ADMINISTER_RECOVERY_LOG or
ADMRL
STATUS=status_variable

Parameters

STATUS

Optional SCL status variable in which the completion status of the command is returned. Its use depends on whether the command is executed in a batch job or an interactive session.

In a batch job, when an error is returned for a subcommand without its own status variable, the error is stored in the ADMINISTER_RECOVERY_LOG status variable, if any, and the session terminates.

In an interactive session, only the completion status of the ADMINISTER_RECOVERY_LOG command and its QUIT subcommand are stored in the ADMINISTER_RECOVERY_LOG status variable. Errors returned by other subcommands are displayed at the terminal and the session continues.

Examples The following is the minimal Administer_Recovery_Log session; it does nothing.

```
/administer_recovery_log
admrl/quit
/
```

BACKUP_LOG Subcommand

Purpose Initiates an immediate backup of the log.

Format

BACKUP_LOG or
BACL
STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the subcommand is returned.

Remarks

- This subcommand must be preceded in the session by a USE_LOG subcommand to specify the log to be backed up.
- This subcommand can be performed only on a log that has been configured for log backups. (This is done using the CONFIGURE_LOG_BACKUP subcommand.)
- This subcommand cannot be used in the same session in which the log is configured. This is because a log that is configured in a session is considered to be a new log, and new logs cannot be backed up.
- This subcommand causes an immediate repository switch which, in turn, initiates a backup of the log.
- You should use this subcommand in both of the following situations:
  - Log users are receiving the status AAE$LOG_TEMPORARILY_FULL, which indicates that an immediate repository switch is needed.
  - A system failure seems imminent.

Examples The following session initiates an immediate repository switch and backup for the existing log in $USER.MY_LOG.

```
/administer_recovery_log
admrl/use_log, catalog=$user.my_log
admrl/backup_log
admrl/quit
/
```

CANCEL_LOG_CHANGES Subcommand

Purpose Discards the log specifications and any delete requests accumulated in the session.

Format

CANCEL_LOG_CHANGES or
CANLC
STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the subcommand is returned.

Remarks

- All accumulated log specifications and delete requests are discarded before they are put into effect by the QUIT subcommand.
- This subcommand is appropriate only after a USE_LOG or DELETE_LOG subcommand has been entered.
- You can begin accumulating log specifications again after this subcommand. To do so, you must begin with another USE_LOG subcommand to specify the log to be created or changed.

Examples The following session enters a change for $USER.MY_LOG, but then discards the change so the session does nothing.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/set_performance_option, emphasis=speed
admrl/cancel_log_changes
admrl/quit
/
```

CLEAR_PROBLEM_JOURNAL Subcommand

Purpose Clears the problem journal for the log.

Format **CLEAR_PROBLEM_JOURNAL** or
CLEPJ
STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the subcommand is returned.

Remarks

- The system maintains a problem journal in each log in which it records any problems that have occurred while using the log.
- You should display the problem journal before clearing it. To do so, use the DISPLAY_PROBLEM_JOURNAL subcommand.
- The log referenced is the log specified on the USE_LOG subcommand earlier in the session.
- Unlike log changes, this subcommand clears the problem journal itself; the request does not wait until the QUIT subcommand is entered. Thus, a request to clear the problem journal cannot be undone by a CANCEL_LOG_CHANGES subcommand.

Examples The following session prints the contents of the problem journal for $USER.MY_LOG before clearing the problem journal.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/display_problem_journal, output=log_problems
admrl/print_file, log_problems
admrl/clear_problem_journal
admrl/quit
/
```

CONFIGURE_LOG_BACKUP Subcommand

Purpose Establishes the backup file pool for the log.

Format

CONFIGURE_LOG_BACKUP or
CONLB
ADD_FILE=file
REMOVE_FILE=file
MEDIA=keyword
EXTERNAL_VSN=list of string
RECORDED_VSN=list of string
TYPE=keyword
VERIFY=boolean
FILE_CLASS=character
INITIAL_VOLUME=name
STATUS=status_variable

Parameters

ADD_FILE or AF

File to be added to the pool of backup files for the log. If ADD_FILE is omitted, no backup file is added.

REMOVE_FILE or RF

File to be removed from the pool of backup files for the log. If REMOVE_FILE is omitted, no backup file is removed.

MEDIA or M

Device class of the file specified by the ADD_FILE parameter.

MAGNETIC_TAPE_DEVICE or MTD

Indicates that the log files are backed up to a labeled tape.

MASS_STORAGE_DEVICE or MSD

Indicates that the log files are backed up to disk. (The next four parameters are not used.)

The default value is MAGNETIC_TAPE_DEVICE.

EXTERNAL_VSN or EVSN

List of external VSNs identifying the tape volumes that compose the file specified by the ADD_FILE parameter. The VSNs are specified as strings of from 1 through 6 characters enclosed in apostrophes. This parameter must be specified if MEDIA is set to MAGNETIC_TAPE_DEVICE.

RECORDED_VSN or RVSN

List of recorded VSNs of the tape volumes that compose the file specified by the ADD_FILE parameter. The recorded VSN is in the ANSI VOL1 label on the volume. The VSNs are specified as strings of from 1 through 6 characters enclosed in apostrophes. This parameter must be specified if MEDIA is set to MAGNETIC_TAPE_DEVICE.

TYPE or T

Tape density written by a nine-track tape drive for the file specified by the ADD_FILE parameter. This parameter is used only if MEDIA is set to MAGNETIC_TAPE_DEVICE.

MT9$800

Indicates 800 cpi.

MT9$1600

Indicates 1600 cpi.

MT9$6250

Indicates 6250 cpi.

The default value is MT9$6250.

VERIFY or V

Indicates whether the backup file specified by the ADD_FILE parameter is verified. This parameter is used only if MEDIA is set to MAGNETIC_TAPE_DEVICE.

TRUE or YES or ON

The magnetic tape is mounted; the backup file is opened to verify that it exists and that it has read and write capabilities.

FALSE or NO or OFF

The backup file is not verified.

The default value is TRUE.

FILE_CLASS or FC

Specifies the class of the file to be assigned. Refer to the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual, Volume 2, for class assignments and a complete description of this parameter. This parameter is used only if MEDIA is set to MASS_STORAGE_DEVICE.

INITIAL_VOLUME or IV

Name specifying the volume serial number (VSN) of the mass storage volume to which the file is to be assigned. The name is specified as a string of from 1 through 6 characters. Refer to the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual, Volume 2, for a complete description of this parameter. This parameter is used only if MEDIA is set to MASS_STORAGE_ DEVICE.

STATUS

Optional SCL status variable in which the completion status of the subcommand is returned.

Remarks

- This subcommand should be preceded in this session by a SET_LOG_BACKUP_ACCOUNT subcommand.
- When a file is backed up to tape, the tape is mounted, validated, and initialized at the time this subcommand is entered.
- A tape backup file must be labeled. Labeling of tapes is done from the system console by the system operator.
- A mass storage backup file must not previously exist; it is created and initialized at the time this subcommand is entered.
- A mass storage backup file is specified by its file path. However, any file cycle specification on the file path is ignored. The backup is always written to cycle 1.

- If any backup files are configured for the log, a backup file must be configured for each log repository. For example, if backup files are configured, a log with five repositories must have five backup files.

- This subcommand can be specified only for a new log. Backup files cannot be configured for an existing log.

- Each CONFIGURE_LOG_BACKUP subcommand can specify one file to be added and one file to be removed from the pool of backup files for the log.

  The files specified by the ADD_FILE and REMOVE_FILE parameters can be the same file. This would be done when changing the description of the backup file.

  - The log referenced by each CONFIGURE_LOG_BACKUP subcommand is the log specified on the USE_LOG subcommand earlier in the session.

  - The backup file pool takes effect when the QUIT subcommand is entered. It is discarded if a CANCEL_LOG_CHANGES subcommand is entered.

  - Once in effect, the backup file pool cannot be modified. If modification is desired, the log must be deleted (using DELETE_LOG) and reconfigured.

  - To see the current backup files, enter a DISPLAY_LOG_CONFIGURATION subcommand. The list reflects the changes made by subcommands in the session, although the changes do not take effect until the QUIT subcommand.

- The FILE_CLASS and INITIAL_VOLUME parameters are described in detail as parameters of the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual.

- If you have established a working catalog, you must enter tape file names as $LOCAL.filename for the ADD_FILE and REMOVE_FILE parameters.

Examples The following session configures the backup files TAPE1, TAPE2, and TAPE3 for log $USER.MY_LOG. Each tape is to consist of two labeled, 6250 cpi tape volumes.

```
/admrl
admrl/usel, $user.my_log
admrl/conlr, repositories=3
admrl/setlba, u=sonya, p=uquiwi
admrl/conlb, tapel, ..
admrl../evsn=('TP1', 'TP2'), rvsn=('X3241','X3242')
admrl/conlb, tape2, ..
admrl../evsn=('TP3', 'TP4'), rvsn=('Y4310','Y4311')
admrl/conlb, tape3, ..
admrl../evsn=('TP5', 'TP6'), rvsn=('Z5011','Z5012')
admrl/quit
/
```

CONFIGURE_LOG_RESIDENCE Subcommand

Purpose Establishes the configuration of the log.

Format

CONFIGURE_LOG_RESIDENCE or
CONLR

REPOSITORIES=integer
REPOSITORY_SWITCHING_SIZE=integer
REPOSITORY_SWITCHING_TIME=integer
SWITCH_SUPPRESSION_SIZE=integer or keyword
SWITCH_SUPPRESSION_TIME=integer or keyword
REPOSITORY_SIZE_LIMIT=integer
FILE_CLASS=character
INITIAL_VOLUME=name
STATUS=status_variable

Parameters

REPOSITORIES or R

Number of disk-resident repositories for the log (integer from 2 through 4096). The default value is 5.

If a backup account and backup pool are specified for the log, the log must have at least 3 repositories.

REPOSITORY_SWITCHING_SIZE or RSS

Repository size threshold for the log (in bytes, from 500,000 through 2,132,483,647 [$(2^{31} - 1) - 15{,}000{,}000$]). The default value is 70,000,000 bytes.

REPOSITORY_SWITCHING_TIME or RST

Repository time threshold for the log (in minutes, from 1 through 525,600 [365 days]). The default value is 1440 (24 hours).

SWITCH_SUPPRESSION_SIZE or SSS

Minimum repository size required before switching (in bytes, from 500,000 through 2,132,483,647 [$(2^{31} - 1) - 15{,}000{,}000$]). The default value is 0.

SWITCH_SUPPRESSION_TIME or SST

Minimum repository time required before switching (in minutes, from 1 through 525,600 [365 days]). The default value is 0.

REPOSITORY_SIZE_LIMIT or RSL

Absolute maximum repository size limit (in bytes, from 15,500,000 through 2,147,483,647 [2^{31} - 1]). It must be at least 15,000,000 bytes larger than the REPOSITORY_SWITCHING_SIZE. The default value is 100,000,000 bytes.

FILE_CLASS or FC

Specifies the class of the file to be assigned. Refer to the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual, Volume 2, for class assignments and a complete description of this parameter.

INITIAL_VOLUME or IV

Name specifying the volume serial number (VSN) of the mass storage volume to which the file is to be assigned. The name is specified as a string of from 1 through 6 characters. Refer to the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual, Volume 2, for a complete description of this parameter.

STATUS

Optional SCL status variable in which the completion status of the subcommand is returned.

Remarks

- For more information on the log configuration parameters, see Configuring a Log earlier in this chapter.
- The FILE_CLASS and INITIAL_VOLUME parameters are described in detail as parameters of the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance manual, Volume 2.
- A repository switch occurs when any of the following events occurs:
  - When the repository size reaches the REPOSITORY_SWITCHING_SIZE for the log, the system checks whether the SWITCH_SUPPRESSION_TIME has passed since the last repository switch. If so, a switch occurs.

  - When the time since a repository switch reaches the REPOSITORY_SWITCHING_TIME for the log, the system checks whether the SWITCH_SUPPRESSION_SIZE has been reached. If so, a switch occurs.

  - When you enter a BACKUP_LOG subcommand.

- During normal log activity, the active repository size should never approach the REPOSITORY_SIZE_LIMIT.

  If the repository size reaches the midpoint between the REPOSITORY_SWITCHING_SIZE and the REPOSITORY_SIZE_LIMIT, no more update operations are allowed for the files using the log. Each update attempt returns the abnormal status AAE$LOG_TEMPORARILY_FULL. This continues until a repository switch occurs.

  You can clear an AAE$LOG_TEMPORARILY_FULL status by entering a BACKUP_LOG subcommand in an Administer_Recovery_Log utility session. The subcommand initiates an immediate repository switch as part of the back process.

- This subcommand can be specified only for a new log. The configuration cannot be changed for an existing log.

Examples The following session creates a new log using only three repositories. Three mass storage files are designated as the backup files. All other log specifications use the default values.

```
/administer_recovery_log
admr1/use_log, $user.my_log
admr1/configure_log_residence, ..
admr1../repositories=3
admr1/set_log_backup_account, user=sonya, ..
admr1../password=uquiwi
admr1/configure_log_backup, ..
admr1../add_file=$user.backup1, ..
admr1../media=mass_storage_device
admr1/conlb, $user.backup2, m=msd
admr1/conlb,$user.backup3, m=msd
admr1/display_log_configuration

Administer_Recovery_Log                    1987-06-05
NOS/VE Keyed File Utilities 1.4 87156       16:36:29

Log_Control_File = :NVE.SONYA.MY_LOG.AAF$LOG_
CONTROL_FILE.1
Number_Of_Repositories        : 3
Current Repository            : 0
Repository_Expiration_Time    :
Repository_Size_Limit         : 100000000 "bytes"
Repository_Switching_Size     : 70000000 "bytes"
Repository_Switching_Time     : 1440 "minutes"
Switch_Suppression_Size       : 0 "bytes"
Switch_Suppression_Time       : 0 "minutes"

Performance_Option
  Parcel                      : Balanced
  Record                      : Balanced
Verify_Log_Entries            : No
```

```
Backup_Account
  User                          : SONYA
  Family_Name                   : UQUIWI
  User_Job_Name                 :
  Job_Class                     :
  Account                       :
  Project                       :
  Output_Disposition            : PRINTER
  User_Information              :

Oldest_Valid_Log_Entry          :

Log_Backup_Pool
  Log_Backup_File_Path          : :NVE.SONYA.BACKUP1.1
                      [ next available log backup file ]

    Media                       : Disk
    Oldest_Entry                :
    Newest_Entry                :
    Damage_Conditions           : None

  Log_Backup_File_Path          : :NVE.SONYA.BACKUP2.1

    Media                       : Disk
    Oldest_Entry                :
    Newest_Entry                :
    Damage_Conditions           : None

  Log_Backup_File_Path          : :NVE.SONYA.BACKUP3.1

    Media                       : Disk
    Oldest_Entry                :
    Newest_Entry                :
    Damage_Conditions           : None

admrl/quit
/
```

DELETE_LOG Subcommand

Purpose Requests deletion of an existing log.

NOTE

Once the log is deleted, any keyed file using the log cannot be updated until either logging for the file is turned off (the LOGGING_OPTIONS attribute of the file includes NONE) or the log is recreated. If the log is recreated, any keyed files using the log must be backed up before entries can be made.

Format

DELETE_LOG or
DELL
CATALOG=file
RETAIN_CONFIGURATION=boolean
STATUS=status_variable

Parameters

CATALOG or C

Catalog path of the log to be deleted. This parameter is required.

RETAIN_CONFIGURATION or RC

Indicates whether the log configuration is kept. This parameter is required.

TRUE or YES or ON

Empty the repositories and the log journal, but keep the log configuration. This has the same effect as deleting the log completely and then recreating it with the same configuration.

FALSE or NO or OFF

Delete all files composing the log, including the repositories, the log journal, and mass storage log backup files.

STATUS

Optional SCL status variable in which the completion status of the subcommand is returned.

Remarks

- The logs specified by DELETE_LOG subcommands are not deleted until the QUIT subcommand is entered for the session. A CANCEL_LOG_CHANGES subcommand clears any pending deletion requests.

- If the log configuration is to be retained, the subcommand deletes all of the log data on the repositories but the log continues to exist and can continue to be used. (You may want to backup your keyed files before deleting the log data; otherwise you are not protected by a recovery log. Be aware that once the log data is deleted, any keyed files using the log must be backed up again. This is because a backup indicator recorded on the log is required for any keyed file that will use the log. By deleting the log data, all past backup indicators are lost.)

  If the log configuration is not to be retained, the subcommand deletes all files relating to the log in the catalog. The catalog is no longer usable as a log until a new log is created in it.

  If all files in the catalog have been deleted, the catalog is deleted as well.

- The catalog used is specified on the DELETE_LOG subcommand. Therefore, the subcommand does not reference the log specified by the USE_LOG subcommand. More than one log can be deleted in a session.

- To change the RETAIN_CONFIGURATION value, re-enter this subcommand with the correction.

Examples The following session requests deletion of log $USER.MY_LOG, but then cancels the request:

```
/administer_recovery_log
admrl/delete_log, $user.my_log, ..
admrl../retain_configuration=false
admrl/cancel_log_changes
admrl/quit
/
```

DISPLAY_LOG_CONFIGURATION Subcommand

Purpose Displays the current log specifications.

Format

DISPLAY_LOG_CONFIGURATION or
DISLC
OUTPUT=file
STATUS=status_variable

Parameters

OUTPUT or O

File to which the display is written.

The file is positioned according to the file position ($BOI, $EOI) appended to the file reference or, if no position is specified, according to its OPEN_POSITION attribute value.

If OUTPUT is omitted, the display is written to the standard output file, $OUTPUT.

STATUS

Optional SCL status variable in which the completion status of the subcommand is returned.

Remarks

- This subcommand must be preceded in the session by a USE_LOG subcommand to specify the log whose configuration is displayed.
- The display includes the following information:

  Number of repositories configured for the log
  Current repository
  Repository expiration time
  Repository size limit
  Repository switching size
  Repository switching time
  Switch suppression size
  Switch suppression time
  Validation information used by backup jobs
  (The password is suppressed.)
  Backup file names and specifications
  Performance option (speed or reliability for records)
  Verification level (checksum or no checksum)
  Oldest valid log entry

Examples The following session begins configuring a log for $USER.MY_LOG, but then cancels the configuration and quits.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/configure_log_residence, repositories=6
admrl/display_log_configuration
Administer_Recover_Log                    1987-06-05
NOS/VE Keyed File Utilities 1.4 87156      15:37:37

Log_Control_File = :NVE.SONYA.MY_LOG.AAF$LOG_
CONTROL_FILE.1

Number_Of_Repositories        : 6
Current_Repository            : 0
Repository_Expiration_Time    :
Repository_Size_Limit         : 100000000 "bytes"
Repository_Switching_Size     : 70000000 "bytes"
Repository_Switching_Time     : 1440 "minutes"
Switch_Suppression_Size       : 0 "bytes"
Switch_Suppression_Time       : 0 "minutes"

Performance_Option
  Parcel                      : Balanced
  Record                      : Balanced
Verify_Log_Entries            : No

Backup_Account
  User                        :
  Family_Name                 :
  Job_Class                   :
  Account                     :
  Project                     :
  Output_Disposition          : PRINTER
  User_Information            :

Oldest_Valid_Log_Entry        :

The Log_Backup_Pool is empty.

admrl/cancel_log_changes
admrl/quit
/
```

DISPLAY_PROBLEM_JOURNAL Subcommand

Purpose Displays the problem journal for the log.

Format

DISPLAY_PROBLEM_JOURNAL or
DISPJ
OUTPUT=file
STATUS=status_variable

Parameters

OUTPUT or O

File to which the display is written.

The file is positioned according to the file position ($BOI, $EOI) appended to the file reference or, if no position is specified, according to its OPEN_POSITION attribute value.

If OUTPUT is omitted, the display is written to the standard output file, $OUTPUT.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- The system records any problems that have occurred while using the log in the problem journal for the log. The problems could be:
  - Checksum errors (if verification is requested)
  - Situations that caused the log to be temporarily full (nonfatal status AAE$LOG_TEMPORARILY_FULL).
  - Partial log entries (indicating log interruptions).
  - Tape loading errors for a log backup file.
  - Tape parity errors for a log backup file.
- The log referenced is the log specified on the USE_LOG subcommand earlier in the session.

Examples The following session writes the problem journal for $USER.MY_LOG to file LOG_PROBLEMS and prints it.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/display_problem_journal, ..
admrl../output=log_problems
admrl/print_file, log_problems
admrl/quit
/
```

HELP Subcommand

Purpose Provides access to online information about the utility.

Format

HELP or
HEL
SUBJECT=string
MANUAL=file
STATUS=status_variable

Parameters

SUBJECT or S

Topic to be found in the index of the online manual. The topic must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands and prompts for display of a subcommand description in the online manual.

MANUAL or M

Online manual file to be read. If you omit the MANUAL parameter, the default is AFM. The subcommand searches for the file in the working catalog and then in the $SYSTEM.MANUALS catalog.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

If STATUS is omitted, the completion status is returned to the terminal in an interactive session or, in a batch job, to the status variable specified on the ADMINISTER_RECOVERY_LOG command, if any.

Remarks

- If the SUBJECT parameter specifies a topic that is not in the manual index, a nonfatal error is returned notifying you that the topic could not be found.
- The default manual file, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for full-screen applications, online manuals are displayed in screen mode. Help on reading online manuals is available in the online manual. To leave the online manual and return to the utility, use QUIT.

Examples The following session shows the default display returned by the HELP subcommand.

```
/administer_recovery_log
admrl/help

The following Administer_Recovery_Log subcommands are
available:

BACKUP_LOG
CANCEL_LOG_CHANGES
CLEAR_PROBLEM_JOURNAL
CONFIGURE_LOG_BACKUP
CONFIGURE_LOG_RESIDENCE
DELETE_LOG
DISPLAY_LOG_CONFIGURATION
DISPLAY_PROBLEM_JOURNAL
HELP
QUIT
SET_LOG_BACKUP_ACCOUNT
SET_PERFORMANCE_OPTION
SET_VERIFICATION_LEVEL
USE_LOG

For a description of a subcommand in the online
manual, enter:

HELP subject = '<subcommand>'

To return from an online manual, enter:

QUIT
admrl/quit
/
```

QUIT Subcommand

Purpose Ends the Administer_Recovery_Log session.

Format

QUIT or
QUI
APPLY_LOG_CHANGES = boolean
STATUS = status_variable

Parameters *APPLY_LOG_CHANGES or ALC*

Indicates whether the log repositories are created or updated based upon the accumulated log specifications.

TRUE or YES or ON

The log is created or updated. Any logs specified on a DELETE_LOG subcommand during the session are deleted.

If a new log is being created, the log catalog is created if it does not exist. The log files are created and initialized. If the log catalog already exists, only the performance option and backup account information can be changed.

FALSE or NO or OFF

The log is not created or updated; log specifications are discarded. Any logs specified on a DELETE_LOG subcommand during the session are kept.

The default value is TRUE.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- To discard the accumulated log specifications and delete requests before ending the session, you can also enter a CANCEL_LOG_CHANGES subcommand before entering the QUIT subcommand.

- The changes specified by the following subcommands do not take effect until the log changes are applied when the QUIT subcommand is entered:

```
CONFIGURE_LOG_BACKUP
CONFIGURE_LOG_RESIDENCE
DELETE_LOG
SET_LOG_BACKUP_ACCOUNT
SET_PERFORMANCE_OPTION
SET_VERIFICATION_LEVEL
```

SET_LOG_BACKUP_ACCOUNT Subcommand

Purpose Specifies the validation information used by backup jobs for the log.

NOTE

Each time the password is changed for the user name used as the backup account, the password must also be changed in the log configuration. Otherwise, all subsequent backup jobs fail to execute.

Format

SET_LOG_BACKUP_ACCOUNT or
SETLBA
USER=name
PASSWORD=name
FAMILY_NAME=name
USER_JOB_NAME=name
JOB_CLASS=name
ACCOUNT=name
PROJECT=name
OUTPUT_DISPOSITION=file or keyword
USER_INFORMATION=string
STATUS=status_variable

Parameters

USER or U

User name under which backup jobs are run. This parameter is required.

PASSWORD or PW

Password for the user name specified by the USER parameter. This parameter is required.

FAMILY_NAME or FN

Optional family name under which backup jobs are run. If FAMILY_NAME is omitted, backup jobs run under the family to which the specified user name belongs.

USER_JOB_NAME or UJN or JOB_NAME or JN

Optional name by which the backup jobs are identified in the system. If USER_JOB_NAME is omitted, the name assigned backup jobs is the user name.

JOB_CLASS or JC

Optional job class in which the backup jobs are run. If JOB_CLASS is omitted, the jobs run in the default job class for the user name.

ACCOUNT or A

Account to which resource usage is charged for the backup jobs. If you omit this parameter for a user name that requires an account, the backup jobs will fail to execute. (See the Remarks.)

PROJECT or P

Project to which resource usage is charged for the backup jobs. If you omit this parameter for a user name that requires a project, the backup jobs will fail to execute. (See the Remarks.)

OUTPUT_DISPOSITION or OD or STANDARD_OUTPUT or SO

Specifies the default for how the backup job's standard output is to be disposed. If omitted, the attribute associated with this parameter does not change.

File name

The standard output is copied to the specified file name at job end.

DISCARD_ALL_OUTPUT or DAO

All output generated by the backup job is to be discarded at job end.

DISCARD_STANDARD_OUTPUT or DSO

Standard output is to be discarded at job end.

LOCAL or L

Any output generated by the backup job is printed at the destination system rather than being returned to the originating user's default output station.

PRINTER or P

Any output generated by the backup job is returned to the originating user's default output station.

WAIT_QUEUE or WQ

Any output generated by the backup job is returned to the originating user's $WAIT_QUEUE subcatalog on the originating system using the user's job name for the file name. If the $WAIT_QUEUE subcatalog does not exist at the time the output files are returned, it is created for the user.

The default value is PRINTER.

USER_INFORMATION or UI

Specifies a user information string of up to 256 characters. This string enables you to pass information (such as a file path) to a backup job. This string is also passed on to all output files generated by the backup job.

If omitted, the user information string associated with the backup job is assumed.

STATUS

Optional SCL status variable in which the completion status of the subcommand is returned.

Remarks

- If backup files are included in the log configuration, each repository switch for the log starts a job to backup the log. Each backup job uses the validation information specified on this subcommand.

- To determine if the ACCOUNT and PROJECT parameters are required and the valid JOB_CLASS values, display the validation information for the user name.

  To display validation information for a user name, use the Administer_User utility with the DISPLAY_USER subcommand. If you are logged in as the family administrator, you can display information on any user in the family; otherwise, you can only display information for the user name you are using.

  The following example shows only the display information that is specified:

```
/administer_user
AV/display_user, sonya, ..
AV../do=(project,project_required,job_class, ..
AV../job_class_defaults)
User = SONYA
```

```
Project              = (ACCTX, PROJY)
Project_Required     = FALSE
Job_Class            = (MAINTENANCE,
                        INTERACTIVE,
                        BATCH,LONG_BATCH,
                        BACKGROUND,
                        FILE_TRANSFER,
                        SHORT_BATCH
Job_Class_Defaults
  Interactive = INTERACTIVE  Batch = BATCH
```

The Project line shows the account and project for the user and the Project_Required line indicates whether their entry is required. The Job_Class line lists the valid job classes and the Job_Class_Defaults lists the default job class for each mode.

For more information on family administration, see the NOS/VE User Validation manual. For more information on user validation, see the NOS/VE System Usage manual.

Examples The following session modifies the $USER.MY_LOG configuration when the password for its backup account has changed.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/set_log_backup_account, user=sonya, ..
admrl../password=newpw
admrl/quit
/
```

SET_PERFORMANCE_OPTION Subcommand

Purpose Specifies the performance emphasis (speed or reliability) for the log.

Format

SET_PERFORMANCE_OPTION or
SETPO
EMPHASIS=keyword
LOG_ENTRY=keyword
STATUS=status_variable

Parameters

EMPHASIS or E

Specifies whether speed or reliability is more important. This parameter is required.

SPEED or S

Speed is more important than reliability.

RELIABILITY or R

Reliability is more important than speed.

BALANCED or B

Both speed and reliability are important.

LOG_ENTRY or LOG_ENTRIES or LE

Indicates the types of log entries to which the specified emphasis applies.

RECORD or R

Record entries.

PARCEL or P

For future implementation.

ALL or A

For future implementation.

The default value is RECORD.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- This subcommand determines how frequently log entries in memory are written to disk. (Its purpose is similar to that of the FORCED_WRITE attribute for keyed files.)
- If this subcommand is not specified, the default performance option is BALANCED.
- The EMPHASIS values have the following meanings:

  SPEED

  The system memory manager determines when log entries are written to disk.

  RELIABILITY

  Each log entry is written to disk before the next log entry begins.

  BALANCED

  The system must begin writing a log entry to disk before the next log entry can begin.

- Any value specified for parcels is recorded for future use, but is currently ignored.

Examples The following session changes the performance options for $USER.MY_LOG.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/set_performance_option, ..
admrl../emphasis=reliability
admrl/quit
/
```

SET_VERIFICATION_LEVEL Subcommand

Purpose Indicates whether checksums should be performed for the header and trailer parts of log records.

Format

SET_VERIFICATION_LEVEL or
SETVL
VERIFY_LOG_ENTRIES=boolean
STATUS=status_variable

Parameters

VERIFY_LOG_ENTRIES or VLE

Indicates whether checksums are performed for the log. This parameter is required.

TRUE or YES or ON

Checksums are performed.

FALSE or NO or OFF

Checksums are not performed.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- This subcommand can be specified only for a new log. The verification level cannot be changed for an existing log.
- This subcommand is optional. If it is omitted from a session that creates a new log, the default verification level is FALSE.

USE_LOG Subcommand

Purpose Establishes the log to be created or changed by the session.

Format

USE_LOG or
USEL
CATALOG=name
STATUS=status_variable

Parameters

CATALOG or C

Catalog path for the log created or changed by the session.

A session can create or change only one log; therefore, any subsequent USE_LOG subcommands are ignored.

If the catalog does not exist, it is created. If the catalog exists, but does not contain a log, a log is created in it. (The catalog and log are actually created at the end of the session when you enter the QUIT subcommand.) If a log exists in the catalog, the session verifies that the log contains the proper characteristics.

This parameter is required.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- You must establish a catalog before any of the other subcommands (except QUIT, DELETE_LOG, HELP, or CANCEL_LOG_CHANGES (after DELETE_LOG)) can be entered.
- Once established, the catalog can only be changed after using CANCEL_LOG_CHANGES.

Examples The following session establishes $USER.MY_LOG as the log to be used. The performance options for $USER.MY_LOG are changed, but then the changes are cancelled and another log is specified.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/set_performance_option emphasis=reliability
admrl/cancel_log_changes
admrl/use_log, $user.my_log_2
admrl/
```

Restore_Log Utility

The Restore_Log utility is the NOS/VE command utility used to restore a damaged log. (For a description of logs, see Maintaining Update Logs in this chapter.) Just as a keyed file can become damaged because of a system failure, one or more of the files composing the log can become damaged. If this happens, none of the keyed files using the damaged log can be updated or restored until the log is restored. Damage to the log can be detected at any time during an open, update, or close operation on a keyed file.

NOTE

It is recommended that the Restore_Log utility be used only by the log administrator. It is a powerful utility that, if used indiscriminately, can cause unnecessary loss of log information.

Since the log consists of a log control file and a pool of repositories (a repository being a log file in which log entries are written), the type and extent of log damage can vary widely. If the damage affects the log control file itself, you must first restore the log control file, then restore the log repositories. If the log control file is not damaged, you can restore the repositories immediately. In any case, complete restoration of the log can never be guaranteed. The extent of damage can be determined by using the RESTORE_REPOSITORIES or VALIDATE_LOG subcommands of the Restore_Log utility.

NOTE

The Restore_Log utility can only be used to restore older repositories (the repositories that are not active) and the log control file if the log was configured for log backups. This is done using the CONFIGURE_LOG_BACKUP subcommand during an Administer_Recovery_Log utility.

Otherwise, if only the active repository is to be replaced, log backups are not required.

A restored log can be put to use depending on the degree of restoration. If the log is completely restored, that is, no recovery information has been lost, the log is available for both recovery operations and for logging further entries.

If, however, recovery information is lost (for example, the active repository is lost, which had not yet been backed up), or the log control file was restored, the log is available only for recovery operations. To begin recording log entries again, you must switch to a different log, or you must delete the log, then recreate it.

Currently, the Restore_Log subcommands are:

Applicable Subcommand	Task
RESTORE_REPOSITORIES	Disables the log, determines the usability of the log, and restores repositories from the backup files.
VALIDATE_LOG	Disables a damaged log and determines the usability of the log.
RESTORE_LOG_CONTROL_FILE	Disables a damaged log and restores the log control file from the backup file.
DELETE_LOG_CONTROL_FILE	Deletes the log control file.
DELETE_REPOSITORIES	Deletes log repositories.
ENABLE_LOG	Enables the log. This makes the log available for general use.
HELP	Provides access to online information about the utility.
QUIT	Ends the session.

Restoring a Log

The following steps outline the process of restoring a log, then recovering a keyed file.

In this example, a keyed file that is protected by the ENABLE_MEDIA_RECOVERY logging option is determined damaged by an error returned during a keyed-file update. The Restore_Log utility is used to restore the log. The Recover_Keyed_File utility is then used to restore the damaged keyed file from the backup file and update it using the restored log.

NOTE

Depending on the type and extent of damage to the log, other error messages may be displayed in this example.

1. Before any recovery begins, stop processing on all keyed files that use the log.

2. As an extra precaution, turn off logging on all files that use the log (to do this, change the LOGGING_OPTIONS attribute to include NONE), and immediately back up all good files that have been updated since the last backup.

3. Before recovering the keyed file, the log must be restored. Begin the Restore_Log session by entering a RESTORE_LOG command and specifying the catalog of the log to be restored. For example:

```
/restore_log $user.my_log
resl/
```

4. If the log control file is damaged, you must restore it from the backup file using the RESTORE_LOG_CONTROL_FILE. For example:

```
resl/restore_log_control_file media=mass_storage_device ..
resl../backup_file=$user.my_log_backup1

--INFORMATIVE  AA 1600--  The attempt to restore the log
control file was successful.

--INFORMATIVE  AA 1675--  The Restore_Repositories subcommand
should now be entered.
```

   The most recently written backup file should be specified first. If RESTORE_LOG_CONTROL_FILE fails, try again specifying the next most recent backup file, and so on.

5. Restore the repository log files by entering the RESTORE_REPOSITORIES subcommand. For example:

```
resl/restore_repositories

--WARNING AA 1650--  The Restore_Repositories subcommand of
Restore_Log was successful but the log control file may be out
of date.

--INFORMATIVE AA 1640-- The last available update found in
the log was at 01/26/88 13:39:09.  Please ensure this is
correct before attempting to use this log for recovery or
resuming the logging of updates.
```

```
-WARNING AA 1680-- Because the active repository or the log
control file was replaced, the entries in the currently
active repository, if any, have been lost.

--WARNING AA 1690-- The log may be enabled by entering the
Enable_Log subcommand of Restore_Log.  The log may then
only be used for recovering keyed files with Recover_Keyed_
File.  Logging may not resume on this log.
```

Once the log is restored, if recovery information is lost (for example, the active repository is lost, which had not yet been backed up), or if the log control file has been restored, the log is available only for recovery operations. To begin recording log entries again, you must switch to a different log, or you must delete the log, then recreate it.

6. Enable the log so that it is available for general use. For example:

```
resl/enable_log

--INFORMATIVE AA 1600-- The attempt to enable the log was
successful.
```

7. Enter the QUIT subcommand to end the session. For example:

```
resl/quit
/
```

8. Begin a Recover_Keyed_File session to restore the keyed file from its latest backup copy. For example:

```
/recover_keyed_file $user.my_ keyed_file
reckf/recover_file_media

--INFORMATIVE AA 1495-- Log $SYSTEM.AAM.AAF$SHARED_
RECOVERY_LOG : Recover_File_Media is attempting to find a
backup of file :NVE.MY_KEYED_FILE.1 in this log.

--WARNING AA 1410-- IMPORTANT - File :NVE.MY_KEYED_FILE.1 :
RECOVER_KEYED_FILE is deleting this file to ensure there is
disk space for the backup file to be reloaded.  To re-start
RECOVER_KEYED_FILE, if it terminates before the file is
successfully restored, you must supply the full file path,
including the cycle number (1).

--WARNING AA 1480-- IMPORTANT - File :NVE.MY_KEYED_FILE.1 :
The cycle number of this file is 1.
```

```
--WARNING AA 1415-- IMPORTANT - The LOG_RESIDENCE should
also be set to :NVE.MY_LOG using the SET_FILE_ATTRIBUTES
command if RECOVER_KEYED_FILE is restarted for this file.

--INFORMATIVE AA 1380-- Recover_Keyed_File is now attempting
to restore :NVE.MY_KEYED_FILE.1 from the backup file.

--INFORMATIVE AA 1495-- Log :NVE.MY_LOG : Recovery_File_
Media is attempting to find a backup of file
:NVE.MY_KEYED_FILE.1 in this log.

--INFORMATIVE AA 1500-- Recover_File_Media has successfully
reloaded file :NVE.MY_KEYED_FILE.1 using the backup record
from log :NVE.MY_LOG.  The date of the backup was
January 26, 1988 at 1:37 PM.

--INFORMATIVE AA 1385-- Recover_Keyed_File is now applying the
changes from the log to the file restored from the backup.

--INFORMATIVE AA 1465-- File :NVE.MY_KEYED_FILE.1 :
Recover_File_Media processed 4 records from the log.  There
were 0 trivial errors.
```

9. Enter the QUIT subcommand to end the session. For example:

```
reckf/quit
/
```

RESTORE_LOG Command

Purpose Begins a Restore_Log utility session.

Format

RESTORE_LOG or
RESL
LOG_RESIDENCE = catalog
STATUS = status_variable

Parameters

LOG_RESIDENCE or LR

Catalog path containing the files composing the log to be restored. This parameter is required.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Only the completion status of the RESTORE_LOG command and its QUIT subcommand are stored in the RESTORE_LOG status variable. Errors returned by other subcommands are displayed at the terminal and the session continues.

Remarks Immediately after entering the Restore_Log session, you should use the VALIDATE_LOG or RESTORE_REPOSITORIES subcommands (described later in this chapter) to determine the type and extent of log damage, if any.

Examples Assuming $USER.MY_LOG is an existing update recovery log, the following is the minimal Restore_Log session; it does nothing.

```
/restore_log, $user.my_log
resl/quit
/
```

RESTORE_REPOSITORIES Subcommand

Purpose Restores damaged repository log files from the log backup files.

Format

RESTORE_REPOSITORIES or
RESR
STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- Older repositories (that is, non-active repositories) can be restored only if the log was configured for automatic backups (see CONFIGURE_LOG_BACKUPS of the Administer_Recovery_Log utility). If the active repository is to be replaced, backups are not required.

- If the log is not already disabled, RESTORE_REPOSITORIES immediately disables it. This is to ensure that the log is not used while it is being restored. Once the log is restored, it can be enabled using ENABLE_LOG (described later in this section).

- Initially, RESTORE_REPOSITORIES determines the usability of the log; that is, the type and extent of log damage, if any. The usability of the log is determined as follows:

  - If damage to the log control file is detected, the following message is displayed:

    ```
    The log control file did not pass
    validation.  It should be reloaded from
    a log backup using the Restore_Log_
    Control_File subcommand.  Unfortunately,
    if there are any entries in the currently
    active repository, they will be lost when
    this command completes.
    ```

    In this case, you must restore the damaged log control file from the backup file using RESTORE_LOG_CONTROL_FILE. Repositories can then be restored using RESTORE_REPOSITORIES.

- If damage to a repository is detected, RESTORE_REPOSITORIES immediately overwrites all existing repositories and restores copies from the backup files.

- If no damage to the log is detected, the following messages are displayed:

```
--INFORMATIVE AA 1640-- The last available
update found in the log was at 01/26/88
1339:09.  Please ensure this is correct
before attempting to use this log for
recovery or resuming the logging of
updates.

--INFORMATIVE AA 1625-- Restore repositories
completed successfully.
```

  Because no damage is detected, no repositories are restored.

  The date and time displayed in the first message indicates the last update in the most recent repository. Thus, up to this time you can restore a damaged keyed file using the Recover_Keyed_File utility.

  Enter ENABLE_LOG before ending the Restore_Log session to enable the log; this makes it available for general use.

- Once the log is restored, if recovery information is lost (for example, the active repository is lost, which had not yet been backed up), or if the log control file has been restored, the log is available only for recovery operations. To begin recording log entries again, you must switch to a different log, or you must delete the log, then recreate it.

Examples The following session replaces all damaged repositories for log $USER.MY_LOG from the backup files:

```
/restore_log $user.my_log
resl/restore_repositories

--WARNING AA 1650-- The Restore_Repositories
subcommand of Restore_Log was successful but the
log control file may be out of date.

--INFORMATIVE AA 1640-- The last available update
found in the log was at 01/26/88 1339:09.  Please
ensure this is correct before attempting to use
this log for recovery or resuming the logging of
updates.

--WARNING AA 1680-- Because the active repository
or the log control file was replaced, the entries
in the currently active repository, if any, have
been lost.

--WARNING AA 1690-- The log may be enabled by
entering the Enable_Log subcommand of Restore_Log.
The log may then only be used for recovering
keyed files with Recover_Keyed_File.  Logging
may not resume on this log.

resl/enable_log
resl/quit
/
```

VALIDATE_LOG Subcommand

Purpose Determines the usability of the log; that is, the type and extent of log damage, if any.

Format

VALIDATE_LOG or
VALL
STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- If damage to the log is detected and if the log is not already disabled, VALIDATE_LOG immediately disables it. This is to ensure that the log is not used while it is being restored. Once the log is restored, it can be enabled using ENABLE_LOG (described later in this section). If no damage to the log is detected, the log is not disabled.

- The usability of the log is determined as follows:

  - If damage to the log control file is detected, the following message is displayed:

    ```
    The log control file did not pass
    validation.  It should be reloaded from
    a log backup using the Restore_Log_
    Control_File subcommand.  Unfortunately,
    if there are any entries in the currently
    active repository, they will be lost when
    this command completes.
    ```

    In this case, you must restore the damaged log control file from the backup file using RESTORE_LOG_CONTROL_FILE.

  - If damage to a repository is detected, the following message is displayed:

    ```
    The log control file is not ready for
    the DAMAGED condition to be reset and
    the log family is not ready for use in
    recovery.  Enter the RESTORE_REPOSITORIES
    subcommand.
    ```

In this case, you must restore copies from the backup files using RESTORE_REPOSITORIES.

- If no damage to the log is detected, the following message is displayed:

```
--INFORMATIVE AA 1630-- No problems were
found during the validation of the log.
```

In this case, the log is left enabled and unchanged.

RESTORE_LOG_CONTROL_FILE Subcommand

Purpose Restores the log control file from the specified log backup file.

Format

RESTORE_LOG_CONTROL_FILE or
RESLCF
MEDIA = keyword
BACKUP_FILE = file
RECORDED_VSN = list of string
EXTERNAL_VSN = list of string
TYPE = keyword
STATUS = status_variable

Parameters

MEDIA or M

Device class of the log backup file to be restored. This parameter is required.

MAGNETIC_TAPE_DEVICE or MTD

Indicates that the log backup file is stored on a labeled tape. (In this case, the BACKUP_FILE parameter is not used.)

MASS_STORAGE_DEVICE or MSD

Indicates that the log backup file specified by the BACKUP_FILE parameter is stored on disk. (In this case, the RECORDED_VSN, EXTERNAL_VSN, and TYPE parameters are not used.)

BACKUP_FILE or BF

The file path name of one of the backup files in the log (previously established by the CONFIGURE_LOG_BACKUP subcommand of the Administer_Recovery_Log utility) to be used for restoring the log control file. This parameter must be specified if MEDIA is set to MASS_STORAGE_DEVICE.

RECORDED_VSN or RVSN

List of recorded VSNs of the tape volumes that compose the log backup file. The recorded VSN is in the ANSI VOL1 label on the volume. The VSNs are specified as strings of from 1 through 6 characters enclosed in apostrophes. This parameter must be specified if MEDIA is set to MAGNETIC_TAPE_DEVICE.

EXTERNAL_VSN or EVSN

List of external VSNs identifying the tape volumes that compose the log backup file. The VSNs are specified as strings of from 1 through 6 characters enclosed in apostrophes. This parameter is used only when MEDIA is set to MAGNETIC_TAPE_DEVICE.

TYPE or T

Tape density of the nine-track tape drive on which the log backup file was written. This parameter is used only when MEDIA is set to MAGNETIC_TAPE_DEVICE.

MT9$800

Indicates 800 cpi.

MT9$1600

Indicates 1600 cpi.

MT9$6250

Indicates 6250 cpi.

The default value is MT9$6250.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- In general, the backup file that was written to most recently is the best one to specify first as the log backup file. If RESTORE_LOG_CONTROL_FILE fails, try again specifying the next most recent backup file, and so on.

- The log control file can be restored only if the log was configured for log backups (see the CONFIGURE_LOG_BACKUP subcommand of the Administer_Recovery_Log utility). A copy of the log control file exists at the front of each log backup file, having been written there as part of the ongoing process of backing up the log.

- If the log control file is not already disabled, RESTORE_LOG_CONTROL_FILE immediately disables it. This is to ensure the log is not used while it is being restored. The log can be enabled using ENABLE_LOG (described later in this chapter).
- RESTORE_LOG_CONTROL_FILE restores a log control file only if it detects damage to the log control file. Damage to the log control file can also be detected by the RESTORE_REPOSITORIES or VALIDATE_LOG subcommands.
- Once a damaged log control file is restored, the log is no longer available for logging entries. The log is available only for recovering keyed files. To begin logging entries again, you must switch to a different log, or you must delete the log whose log control file has been restored, then recreate it.

Examples The following session replaces the log control file and all damaged repositories for log $USER.MY_LOG from the backup files:

```
resl/restore_log_control_file ..
resl/media=mass_storage_device ..
resl../backup_file=$user.my_log_backup1

--INFORMATIVE AA 1600-- The attempt to restore
the log control file was successful.

--INFORMATIVE AA 1675-- The Restore_Repositories
subcommand should now be entered.
```

```
resl/restore_repositories

--WARNING AA 1650-- The Restore_Repositories
subcommand of Restore_Log was successful but the
log control file may be out of date.

--INFORMATIVE AA 1640-- The last available update
found in the log was at 01/26/88 13:39:09.  Please
ensure this is correct before attempting to use
this log for recovery or resuming the logging of
updates.

--WARNING AA 1680-- Because the active repository
or the log control file was replaced, the entries
in the currently active repository, if any, have
been lost.

--WARNING AA 1690-- The log may be enabled by
entering the Enable_Log subcommand of Restore_Log.
The log may then only be used for recovering keyed
files with Recover_Keyed_File.  Logging may not
resume on this log.

resl/enable_log

--INFORMATIVE AA 1600-- The attempt to enable the log
was successful.

resl/quit
/
```

DELETE_LOG_CONTROL_FILE Subcommand

Purpose Deletes the log control file.

Format **DELETE_LOG_CONTROL_FILE** or
DELLCF
STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the command is returned.

Remarks The log control file should be deleted only if it is damaged or if you want to force the log control file to be restored from the backup file. Damage to the log control file can be detected by the VALIDATE_LOG, RESTORE_REPOSITORIES, or RESTORE_LOG_CONTROL_FILE subcommands.

DELETE_REPOSITORIES Subcommand

Purpose Deletes log repositories.

Format

DELETE_REPOSITORIES or
DELR
REPOSITORIES=list of range of integer or ALL
STATUS=status_variable

Parameters

REPOSITORIES or R

Specifies which repositories in the log are to be deleted. This parameter is required.

List of integer

Specifies the repositories to be deleted. Values can be a list of repository numbers specified in the repository name. Repositories have names in the format AAF$REPOSITORY_n where n is the integer value specified; that is, AAF$REPOSITORY_1, starting at one for the first repository, and incremented sequentially and contiguously. The last repository is specified as AAF$REPOSITORY_0. You can specify as many values as there are repositories to be deleted. If more than one value is specified, the values must be enclosed in parentheses and separated by commas or spaces.

ALL or A

All repositories in the log are deleted.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks Repositories should be deleted only if they are damaged or if you want to force the repositories to be restored from the backup files. Damage to repositories can be detected by the VALIDATE_LOG or RESTORE_REPOSITORIES subcommands.

ENABLE_LOG Subcommand

Purpose Enables a disabled log; that is, makes the log available for general use.

Format

ENABLE_LOG or
ENAL
STATUS=status_variable

Parameters

STATUS

Optional SCL status variable in which the completion status of the command is returned.

Remarks

- If the log is disabled and it is usable; that is, the log is undamaged, ENABLE_LOG enables it. This makes the log available for general use.
- If the log is disabled but not usable, an error is displayed and the log remains disabled. Damage can be detected on the log control file and/or the repositories as follows:
  - If damage to the log control file is detected, the following message is displayed:

```
The log control file did not pass
validation.  It should be reloaded from
a log backup using the RESTORE_LOG_
CONTROL_FILE subcommand.  Unfortunately,
if there are any entries in the currently
active repository, they will be lost when
this command completes.
```

    In this case, you must restore the damaged log control file from the backup file using RESTORE_LOG_CONTROL_FILE.

  - If damage to a repository is detected, the following message is displayed:

```
The log control file is not ready for the
DAMAGED condition to be reset and the log
family is not ready for use in recovery.
Enter the RESTORE_REPOSITORIES subcommand.
```

In this case, you must restore repositories from the backup files using RESTORE_REPOSITORIES.

- If no damage to the log is detected, the following message is displayed:

```
--INFORMATIVE AA 1630-- No problems were
found during the validation of the log.
```

In this case, the log is left enabled and unchanged.

- After enabling the log, and if the active repository and the log control file were not restored, you can turn on logging for the log (if it was previously turned off), then backup the keyed file.
- A log must be enabled and usable before you can use it to recover keyed files.

HELP Subcommand

Purpose Provides access to online information about the utility.

Format

HELP or
HEL
SUBJECT=string
MANUAL=file
STATUS=status_variable

Parameters

SUBJECT or S

Topic to be found in the index of the online manual. The topic must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands and prompts for display of a subcommand description in the online manual.

MANUAL or M

Online manual file to be read. If you omit the MANUAL parameter, the default is AFM. The subcommand searches for the file in the working catalog and then in the $SYSTEM.MANUALS catalog.

STATUS

Optional SCL status variable in which the completion status of the command is returned.

If STATUS is omitted, the completion status is returned to the terminal, to the status variable specified on the RESTORE_LOG command, if any.

Remarks

- If the SUBJECT parameter specifies a topic that is not in the manual index, a nonfatal error is returned notifying you that the topic could not be found.
- The default manual file, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for full-screen applications, online manuals are displayed in screen mode. Help on reading online manuals is available in the online manual. To leave the online manual and return to the utility, use QUIT.

Examples The following session shows the default display returned by the HELP subcommand.

```
/restore_log
resl/help

The following Restore_Log subcommands are available:

VALIDATE_LOG
RESTORE_REPOSITORIES
RESTORE_LOG_CONTROL_FILE
DELETE_REPOSITORIES
DELETE_LOG_CONTROL_FILE
ENABLE_LOG
HELP
QUIT

For the description of a subcommand in the online
manual, enter:  HELP subject = '<subcommand>'

To return from an online manual, enter:  QUIT
resl/quit
/
```

QUIT Subcommand

Purpose Ends the Restore_Log session.

Format

QUIT or
QUI
STATUS=status_variable

Parameters *STATUS*

Optional SCL status variable in which the completion status of the command is returned.

If STATUS is omitted, the completion status is returned to the terminal, to the status variable specified on the RESTORE_LOG command, if any.

Remarks The QUIT command is required to end a session.

Part III: FMU

Introducing FMU 10

Introducing FMU 10

The File Management Utility (FMU) is a general-purpose data reformatting tool. FMU reads records from an input file and reformats the records according to your specifications. FMU can change the order, contents, and representation of record fields. A single FMU run can write multiple output files and can format the records differently for each file.

FMU can perform the following tasks:

- Selectively reformat and reorder data fields
- Selectively convert data to another data type
- Add sequence numbers to records
- Format a file for printing

It can also be used to migrate files from other systems as described in the migration manuals listed in appendix B.

Performance Considerations

As you would expect, the speed of an FMU run depends on the amount of work to be performed by the run. For example, a simple file copy runs faster than a complicated file reformatting for several output files. This section highlights the FMU specifications that increase the work performed by an FMU run.

Directive parsing

An FMU command to copy a file to one output file should use the INPUT and OUTPUT parameters, not directives. Better yet, when appropriate, use the COPY_FILE command to perform a byte-by-byte copy instead of the record-by-record copy performed by FMU.

Print file formatting

Specify a SET_PRINT_ATTRIBUTES directive only if needed. The print formats vary from fastest to slowest in this order: 1, 2, 3, DUMP.

Filling in unassigned output fields

Using NO_PRESET as the RECORD_PRESET_VALUE parameter value specifies less work than a value that requires assignment of values to unassigned fields.

Iterative statements (FOR, LOOP, REPEAT, WHILE)

Processing of an interative statement requires a test and compare for each iteration of the loop. This takes more time than the equivalent sequence of assignment statements.

Expression evaluation

Use the simplest possible expressions; do not use unnecessary function references.

Data type conversion and assignment

The time required to convert data types depends on the complexity of the data type. The data types in order from simplest to most complex are: A I H J P Y Z U Q L F N B G.

Field descriptors that specify a trailing position

When possible, specify the field length in the descriptor so that FMU does not need to calculate the length from the starting and trailing position values.

FMU Command and Directives 11

FMU use requires three steps:

1. Describe the input and output files. (For NOS/VE files, use the SET_FILE_ATTRIBUTES command.)

2. Enter FMU directives in a file. The directives identify the input file and one or more output files and the desired data reformatting.

3. Execute the FMU command with appropriate parameters.

This chapter first describes the commands required to describe the files used by FMU. It then describes the FMU command itself. Finally, it describes the directives that can be stored in an FMU directive file.

Describing NOS/VE Files

This section describes the NOS/VE commands for setting, displaying, and changing file attributes.

SET_FILE_ATTRIBUTES Command

To specify file attribute values for a NOS/VE file, use the NOS/VE SET_FILE_ATTRIBUTES command. You specify the file reference on the SET_FILE_ATTRIBUTES command followed by one or more attribute parameters.

For example, this command specifies indexed-sequential file attributes for your new permanent file MYFILE:

```
set_file_attributes $user.myfile ..
file_organization=indexed_sequential ..
maximum_record_length=150 ..
minimum_record_length=125 ..
key_length=5
```

The complete description of the SET_FILE_ATTRIBUTES is in the NOS/VE Commands and Functions manual. Use of the SET_FILE_ATTRIBUTES command to describe keyed files is described in chapter 6 of this manual. Use of the SET_FILE_ATTRIBUTES command to describe sequential and byte-addressable files is described in the NOS/VE System Usage manual.

DISPLAY_FILE_ATTRIBUTES Command

To see the attribute values defined for a NOS/VE file, enter the DISPLAY_FILE_ATTRIBUTES command specifying the file reference and one or more attributes to be displayed.

For example, this command requests display of the maximum_record_length attribute for your permanent file MYFILE:

```
display_file_attributes $user.myfile ..
display_option=maximum_record_length
```

The complete description of the DISPLAY_FILE_ATTRIBUTES is in the NOS/VE Commands and Functions manual.

CHANGE_FILE_ATTRIBUTES Command

A SET_FILE_ATTRIBUTES command cannot change some attributes after the file has been opened. (If there is data in the file, the file has been opened.)

Some attributes that cannot be changed by a SET_FILE_ATTRIBUTES command can be changed by a CHANGE_FILE_ATTRIBUTES command. The complete description of the CHANGE_FILE_ATTRIBUTES command is in the NOS/VE Commands and Functions manual.

The FMU Command

To execute FMU, you enter the FMU command. This section describes the FMU command and how it is used.

The FMU command is entered like any other NOS/VE command. All FMU parameters are optional. The command has two formats.

This format is used when a directive file is needed:

FILE_MANAGEMENT_UTILITY or
FMU
DIRECTIVES=file
LIST=file
ERROR_DISPOSITION=keyword
STATUS=status_variable

This format performs simple file copying:

FILE_MANAGEMENT_UTILITY (FMU)
INPUT=file
OUTPUT=file
LIST=file
ERROR_DISPOSITION=keyword
STATUS=status_variable

These are the parameter descriptions for the FMU command:

DIRECTIVES (DIR)

File from which FMU reads directives. If you omit the INPUT and OUTPUT parameters, you must specify the DIRECTIVES parameter. If you specify the DIRECTIVES parameter, you must omit the INPUT and OUTPUT parameters.

INPUT (I)

File to be copied. If you specify this parameter, you must omit DIRECTIVES (or set DIRECTIVES=$NULL) and must specify the OUTPUT parameter. The default for INPUT is $INPUT.

OUTPUT (O)

File to which the input file is copied. When specifying this parameter, you must omit DIRECTIVES (or specify DIRECTIVES=$NULL) and must specify the INPUT parameter. The default for OUTPUT is $OUTPUT.

LIST (L)

File to receive the summary of the FMU run, including diagnostic messages. If you omit this parameter, LIST=$LIST is assumed. In an interactive session, the default connection for $LIST is $NULL which discards the listing.

ERROR_DISPOSITION (ED)

Indicates whether FMU aborts if an output file is closed prematurely due to an error.

ABORT (A)	FMU aborts.
NO_ABORT (NA)	FMU continues writing the other output files.

If you omit this parameter, FMU aborts when an output file aborts.

STATUS

Optional SCL status variable in which the FMU completion status is returned. Specifying a status variable allows you to test for error conditions. To use this parameter, you need to have previously declared an SCL STATUS variable.

If you use STATUS, the FMU task does not abort and does not go to a WHEN/WHENEND condition handler. Instead, an error that would abort the task causes FMU to terminate. The status variable is set to the error condition code. See the NOS/VE System Usage manual for an explanation of the STATUS variable and condition handling.

Using FMU to Copy a File

File copying is usually performed by the NOS/VE COPY_FILE or COPY_KEYED_FILE commands. However, the FMU command can also copy a file.

For example, this command copies file FILE1 to file FILE2.

```
fmu input=file1 output=file2 list=list
```

A file copy can copy data to a file whose file attributes differ from those of the input file.

Using the FMU Directive File

To use FMU directives, you must enter the directives in a file and specify the file on the DIRECTIVES parameter.

For example, this command specifies DIRECTIVES_FILE as the file from which directives are read:

```
fmu, directives=directives_file, list=listing_file
```

The input and output files are identified by the SET_INPUT_ATTRIBUTES and SET_OUTPUT_ATTRIBUTES directives in the directives file. (File attributes can be specified, as always, by SET_FILE_ATTRIBUTES or CHANGE_FILE_ATTRIBUTES commands.)

FMU Directives

Generally, when you use FMU, you use FMU directives. You enter the directives in a directive file and specify the file on the DIRECTIVES (DIR) parameter of the FMU command. Not all of the directives are required at all times.

The following is a summary of the FMU directives.

Directive	Abbreviation	Function
SET_INPUT_ ATTRIBUTES	SETIA	Specifies the input file
SET_OUTPUT_ ATTRIBUTES	SETOA	Specifies the output file
SET_SEQUENCE_ ATTRIBUTES	SETSA	Adds record sequence numbers
SET_PRINT_ ATTRIBUTES	SETPA	Specifies print format
CREATE_OUTPUT_ RECORD	CREOR	Marks the beginning of a list of statements that define the record reformatting for an output file
CREATE_OUTPUT_ RECORD_END	CREOREND	Marks the end of the data reformatting statement list

The detailed descriptions of these directives follow the next sections giving general information about directive specification.

Directive Order

You can specify the directives in any order except:

- SET_INPUT_ATTRIBUTES must precede all other directives.
- SET_OUTPUT_ATTRIBUTES for a given output file must precede all other directives for that file.

When more than one output file is used, FMU writes to the output files in the order that their corresponding SET_OUTPUT_ATTRIBUTES directives are specified in the directives file.

Order of Processing

During file processing, each input record is processed by all directives before the next input record is read. FMU processes each input record as follows:

1. Reads a record from the file specified by the FILE parameter of the SET_INPUT_ATTRIBUTES directive.
2. Creates the current output record. The CREATE_OUTPUT_RECORD directive (if specified for the output file) is used to control this process.
3. Sequences the current output record as specified by the SET_SEQUENCE_ATTRIBUTES directive for the output file (if any).
4. Formats the current output record for printing as specified by SET_PRINT_ATTRIBUTES directive for the output file (if any).
5. Writes the current output record to the output file specified by the SET_OUTPUT_ATTRIBUTES directive.

FMU executes multiples of the same directive types in the order that their associated SET_OUTPUT_ATTRIBUTES directives are specified in the directive file.

FMU checks the syntax of all directives in the directive file before it begins file processing. Any diagnostics pertaining to the directives are written to the FMU list file. (For more information on FMU diagnostics, see appendix G.)

FMU gives control back to you when the input file has been read completely and execution terminates.

FMU Directive Format

An FMU directive consists of a directive name followed by a parameter-list. The directive name and the parameter-list are separated by a comma or space. The parameter-list is one or more parameters with one or more spaces or a comma separating each pair of parameters.

For the CREATE_OUTPUT_RECORD directive, the list of parameters is followed by a list of statements, which is terminated by a CREATE_OUTPUT_RECORD_END directive.

A directive can begin in any column and span more than one line. A continuation is indicated by two or more periods (..) at the end of a line.

Here is an example of a set of directives in a directive file:

```
SET_INPUT_ATTRIBUTES, FILE=infile
SET_OUTPUT_ATTRIBUTES, FILE=outfile, ERROR_DISPOSITION=NO_ABORT
    SET_SEQUENCE_ATTRIBUTES, FILE=outfile, SEQUENCE_FIELD=Z[1,3]
    SET_PRINT_ATTRIBUTES, FILE=outfile, PRINT_TITLE='June
      Printout'
    CREATE_OUTPUT_RECORD, FILE=outfile, ..
      RECORD_PRESET_VALUE=CHARACTER_BLANK;
        IF A[5,2] = 'XX' THEN
            A[12,4] = A[17,4]
        IFEND;
    CREATE_OUTPUT_RECORD_END
```

FMU Directive Parameters

You can specify directive parameters according to their position or independent of their position, or both.

Parameters of directives can be expressed in the format:

parameter_name=value

With this format, you can specify the parameters in any order. Parameters can be separated by commas or blanks.

An example of parameters expressed in this format is:

```
SET_OUTPUT_ATTRIBUTES, FILE=myfile, ..
  ERROR_DISPOSITION=NO_ABORT
```

You can specify a parameter value without its parameter name. When doing so, however, you must be sure that the parameter keeps its relative position in the parameter list. Omitted parameters are indicated by additional commas (one for each omitted parameter). The parameter order is shown in the directive description.

The previous SET_OUTPUT_ATTRIBUTES directive can be entered in position-dependent form as follows:

```
SET_OUTPUT_ATTRIBUTES, myfile, , NO_ABORT
```

Directives can have both position-independent and position-dependent parameters, as shown in this example:

```
SET_OUTPUT_ATTRIBUTES, myfile, ERROR_DISPOSITION=NO_ABORT
```

DUPLICATE_SPECIFICATION Parameter

All directives, except SET_INPUT_ATTRIBUTES, have a DUPLICATE_SPECIFICATION (DS) parameter. This parameter enables you to request an identical directive specification for more than one output file without having to repeat parameters or statements.

The format of any directive using the DUPLICATE_SPECIFICATION parameter is:

directive_name, FILE=lfn1, DUPLICATE_SPECIFICATION=lfn2

directive_name

Name of the directive; cannot be SET_INPUT_ATTRIBUTES.

FILE=lfn1

File for which the directive is to be specified.

DUPLICATE_SPECIFICATION=lfn2

File for which the directive specifications to be copied have been previously specified.

The impact of the DUPLICATE_SPECIFICATION parameter can best be seen in its use with the CREATE_OUTPUT_RECORD directive, which can have long statement lists.

For example, consider the following directive sequence:

```
SET_INPUT_ATTRIBUTES  infile
SET_OUTPUT_ATTRIBUTES  outfile1
SET_OUTPUT_ATTRIBUTES  outfile2
CREATE_OUTPUT_RECORD outfile1, ..
 RECORD_PRESET_VALUE=CHARACTER_BLANK
    IF A[3,8] = 'SJCODE' THEN
        A[7, ] = A[13, ]
        A[15,25] = A[22,25]
        A[41,8] = A[52,8]
    IFEND
CREOREND
CREATE_OUTPUT_RECORD, outfile2, DUPLICATE_SPECIFICATION=outfile1
CREOREND
```

In the example, information is taken from the input file, reformatted, and placed on the output files OUTFILE1 and OUTFILE2.

When the DUPLICATE_SPECIFICATION parameter is used, the MACHINE_FORMAT (MF) parameter value must be the same for both output files.

Basic Directive Elements

This section lists the the rules for constructing FMU directives.

Permitted Characters

For constructing a directive, FMU recognizes the following characters:

Lowercase letters a through z
Uppercase letters A through Z
Digits 0 through 9
Special characters + - ( ) = blank , ' ; " . [ ]

In names, FMU recognizes lowercase and uppercase letters, digits 0 through 9, and special characters _ $ # @.

Lowercase letters and uppercase letters are equivalent when used in directives (including directive names, keywords, and statements).

Delimiters

The following characters can be used as delimiters:

'	Delimits literals.
"	Delimits comments.
( )	Encloses some parameter values; encloses position including bit offset; encloses relational expressions; encloses function parameter.
[ ]	Encloses field descriptor parameters.
,	Separates parameters.
;	Terminates a statement or separates statements; terminates directives.
..	Continues statement or directive to next line; can be more than two periods.
blank	Separates parameters within directives.
=	Separates a keyword from its associated parameter value; is the assignment operator used in the CREATE_OUTPUT_RECORD directive assignment statements.

Comments in Directives

You can place comments anywhere, except within literals.

Comments are delimited by a quotation mark ("). The end of a line also terminates a comment.

Names in Directives

A name is a string of from one to 31 alphanumeric characters (also $ # @ _), the first of which must not be a digit. A name must be delimited at both ends by a space, comment, graphic operator, delimiter, or beginning or end of line. A user-supplied name can be a file name or a variable name.

Although NOS/VE allows use of the international characters ([ ] { } \ | ` " ^) in names, FMU does not. If the name of a file to be used contains an international character, attach the file with a local file name valid for FMU, if possible. Otherwise, use COPY_FILE to copy the file to a file with a name valid for use by FMU.

You should not create a name that uses the $ character because it may already be a CDC-defined name.

Integer Constants

Integer constants are unsigned and can be specified as parameter values to directives, functions, field descriptors, and FOR statements.

An integer constant has the form:

n[n...]

Literals

A literal is a string of characters that represents an actual value; the value can be alphanumeric, numeric, or logical.

Literals are represented by a string of alphanumeric characters enclosed by apostrophes. The delimiting apostrophes are excluded from the literal. An apostrophe can be represented in the literal by two successive apostrophes between the delimiting apostrophes.

A literal must not exceed 256 characters.

Strings

ASCII string data can be represented by literals. Blanks are significant and character case (lowercase or uppercase) is preserved in ASCII string literals.

Numeric Literals

In numeric literals, blanks are ignored. FMU performs required character-to-numeric conversion based on type and context of associated item specifications. The notation for representing numbers is one of the following:

```
' [+] n [n...] [.] [n...] [E[ + ] n [n...] ] '
' [+] n [n...] [.] [n...] [D[ + ] n [n...] ] '
```

where n is a digit.

You can also use this form:

```
' [ + ] n [n...] [(base)] '
```

where the base can be 2, 8, 10, or 16. The default base is 10.

For example, the following relational expressions are equivalent:

```
N[1,6] = '16'
N[1,6] = '16(10)'
N[1,6] = '10(16)'
N[1,6] = '20(8)'
```

Logical Data

An alphabetic literal can represent a logical value (data type L) for storage into an L field of an output record.

If the leftmost nonblank character in the literal string is T or t, the L field is set to the binary equivalent of TRUE.

If the leftmost nonblank character of the literal string is F or f, the L field is set to the binary equivalent of FALSE.

If the leftmost nonblank character is a period (.), the next character must be a T, t, F, or f, and is processed as previously described. A field with all blanks is processed as FALSE.

For all other cases, an error occurs.

SET_INPUT_ATTRIBUTES Directive

Purpose Specifies the single input file that is required for an FMU run.

Format

SET_INPUT_ATTRIBUTES (SET_INPUT_ATTRIBUTE or
SETIA)
FILE=local_file_name
STARTING_FILE_POSITION=(integer, keyword1, keyword2)
MAXIMUM_FILE_UNITS=(integer, keyword)
MACHINE_FORMAT=keyword

Parameters

FILE (F)

Local file name of the file in the working catalog containing input records. This parameter is required.

STARTING_FILE_POSITION (SFP)

Value set specifying the file repositioning to be performed. If you omit the STARTING_FILE_POSITION parameter, the default is no repositioning.

The value set is three values enclosed in parentheses. The first value is an unsigned integer specifying the number of units to be skipped; the default is 1.

The second value is one of the following keyword values specifying the type of unit skipped.

RECORDS (RECORD or R)	Skips records (default)
PARTITIONS (PARTITION or P)	Skips partitions

The third value is one the following keyword values specifying the skip direction.

FORWARD (F)	Skips forward (default)
BACKWARD (B)	Skips backward

Be careful when using a default value in a parameter value set. Possible variants include SFP=n, SFP=(,u), SFP=(,,d), SFP=(n,u), SFP=(n,,d), and SFP=(,u,d).

MAXIMUM_FILE_UNITS (MFU)

Value set indicating the maximum number of file units to be processed. If you omit the MAXIMUM_FILE_UNITS parameter, no limit is set; all records in the file are processed.

The value set is two values enclosed in parentheses. The first value is an unsigned integer indicating the file unit limit; the default is 1.

The second value is a keyword value indicating the unit type as follows:

RECORDS (RECORD or R)	The limit is in records (default).
PARTITIONS (PARTITION or P)	The limit is in partitions.

Be careful when using a default value in a parameter value set. Possible variants include MFU=n and MFU=(,u).

MACHINE_FORMAT (MF)

The system that wrote the file.

C180	NOS/VE (default)
C170	NOS or NOS/BE
C7600	SCOPE 2
IBM	IBM
VAX	VAX/VMS (8-bit floating point exponent)
VAXG	VAX/VMS (11-bit floating point exponent)

Examples This directive specifies the FMU input file as INFILE.

```
SET_INPUT_ATTRIBUTES FILE=infile
```

This directive specifies the FMU input file as INFILE. It also specifies the starting file position as 1 record forward and the maximum file units as 50 records.

```
SETIA infile STARTING_FILE_POSITION=1, ..
  MAXIMUM_FILE_UNITS=50
```

SET_OUTPUT_ATTRIBUTES Directive

Purpose Declares an output file to be written.

Each output file requires a SETOA directive. FMU writes a record to each output file in the order that you specify the SETOA directives.

Format

SET_OUTPUT_ATTRIBUTES or
SET_OUTPUT_ATTRIBUTE or
SETOA
FILE=local_file_name
DUPLICATE_SPECIFICATION=local_file_name
ERROR_DISPOSITION=keyword
PARTITION_DISPOSITION=keyword
MAXIMUM_FILE_UNITS=(integer, keyword)
MACHINE_FORMAT=keyword
CONDITION_COLLATING_SEQUENCE=keyword
EXCEPTION_RECORDS_FILE=file
CONVERSION_ERROR_DISPOSITION=keyword

Parameters

FILE or F

Local file name of the file in the working catalog to contain output records. This parameter is required. If no output is to be written, specify $NULL.

DUPLICATE_SPECIFICATION or DS

Local file name whose SET_OUTPUT_ATTRIBUTES directive specifications are to be duplicated for this file (see Duplicating Directive Specifications later in this chapter).

ERROR_DISPOSITION or ED

Determines whether generation of the output file is aborted if an error occurs at execution time.

ABORT or A	Abort on error (default)
NO_ABORT or NA	Do not abort on an error

Aborting on an error means that if an error occurs, the output file is closed, and processing for this file ceases.

If NO_ABORT is specified, formatting of the output file continues, but the record that causes the error is discarded.

PARTITION_DISPOSITION or PD

Causes partition boundaries to be included or excluded. This parameter is applicable only when the input and output files are sequential; otherwise, it is ignored.

INCLUDE_PARTITION or IP	Include partition boundaries (default)
EXCLUDE_PARTITION or EP	Exclude partition boundaries

MAXIMUM_FILE_UNITS or MFU

Value set indicating the maximum number of file units to be written to this output file. If you omit the MAXIMUM_FILE_UNITS parameter, no limit is set; all records are written.

The value set is two values enclosed in parentheses. The first value is an unsigned integer indicating the file unit limit; the default is 1.

The second value is a keyword value indicating the unit type as follows:

RECORDS or RECORD or R	The limit is in records (default).
PARTITIONS or PARTITION or P	The limit is in partitions.

Be careful when using a default value in a parameter value set. Possible variants include MFU=n and MFU=(,u).

MACHINE_FORMAT or MF

The machine format in which the file is to be written.

C180 NOS/VE (default)
C170 NOS or NOS/BE

Use of the MACHINE_FORMAT parameter is described in the migration manuals. (The migration manuals are listed in appendix B of this manual.)

CONDITION_COLLATING_SEQUENCE or CCS

Optional collating sequence to be used when FMU compares or searches for string values when reformatting records for this output file. See the following Remarks for the methods of specifying a collating sequence.

If you omit this parameter, the default ASCII collating sequence is used. (The default differs when migrating files; for details, see the appropriate migration manual.)

EXCEPTION_RECORDS_FILE or EXCEPTION_RECORD_FILE or ERF

File in the working catalog to which records that cannot be written to the output file are written, The record is written when an error is found during CREATE_OUTPUT_RECORD processing of the record.

If this directive specifies ERROR_DISPOSITION=NO_ABORT, each record in error is written to the exception records file. Otherwise, if the ERROR_DISPOSITION parameter is omitted or specifies ABORT, no more than one record is written to the file because the first record error terminates the processing of the output file.

The default exception records file is $NULL, in which case, the records in error are discarded.

CONVERSION_ERROR_DISPOSITION or CED

Action to be taken when a conversion error occurs because the source field contains unrecognizable data.

ABORT or A — No recovery attempt is made; a record formatting error is returned (default).

RECOVER or R — FMU attempts to recover from the error by using the default value for the source field as follows:

0 (zero)	Numeric fields
Spaces	Character (A) fields
FALSE	Logical (L) fields

Remarks The following remarks describe the methods of specifying a collating sequence on the CONDITION_COLLATING_SEQUENCE parameter.

Predefined Collating Sequences

You can specify a predefined collating sequence on the CONDITION_COLLATING_SEQUENCE parameter of the SET_OUTPUT_ATTRIBUTES directive using one of these keywords:

Keyword	Collating Sequence
ASCII or A	Default ASCII collating sequence or as the character set is listed in appendix C
ASCII6_FOLDED or AF	OSV$ASCII6_FOLDED
ASCII6_STRICT or AS	OSV$ASCII6_STRICT
COBOL6_FOLDED or CF	OSV$COBOL6_FOLDED
COBOL6_STRICT or CS	OSV$COBOL6_STRICT
DISPLAY63_FOLDED or DF3	OSV$DISPLAY63_FOLDED
DISPLAY63_STRICT or DS3	OSV$DISPLAY63_STRICT
DISPLAY64_FOLDED or DF4	OSV$DISPLAY64_FOLDED
DISPLAY64_STRICT or DS4	OSV$DISPLAY64_STRICT
EBCDIC or E	OSV$EBCDIC
EBCDIC6_FOLDED or EF	OSV$EBCDIC_FOLDED
EBCDIC6_STRICT or ES	OSV$EBCDIC_STRICT

Lowercase and Uppercase Equal

This manual does not contain listings for the following two collating sequences. These sequences are the same as the default ASCII collating sequence listed in appendix C except that uppercase and lowercase letters are collated equally.

Keyword	Collating Sequence
LOWER_TO_UPPER or LTU	Lowercase collated as uppercase
UPPER_TO_LOWER or UTL	Uppercase collated as lowercase

Indexed-Sequential File

FMU can also use the collation table stored with the input file or with this output file. (A collation table is stored only for an indexed-sequential file with a collated primary key.)

Keyword	Collating Sequence
INPUT_FILE or IF	Input file collation table
OUTPUT_FILE or OF	Output file collation table

A collation table to be stored with a file is specified as the COLLATE_TABLE_NAME file attribute before the first open of the file. The stored collating sequence can be displayed using a DISPLAY_KEYED_FILE_PROPERTIES command.

User-Defined Collating Sequence

You can define a new collating sequence on the CONDITION_COLLATING_SEQUENCE parameter. The collating sequence is specified by a sequence of values enclosed in parentheses. Each value specifies the characters assigned to that position in the collating sequence.

A value can be a string of one or more characters or a range of characters or two or more strings or ranges enclosed in parentheses.

A range is specified by two characters separated by two periods (such as A..Z). The range specifies all characters between, and including, the two characters in the ASCII character set. A backwards range (such as Z..A) is valid.

Each character can be specified by its graphic enclosed in apostrophes (such as 'A') or by its integer character code (such as 65).

Any omitted characters are assigned to the last position of the sequence. No character can appear more than once.

Consider the following example:

```
CONDITION_COLLATING_SEQUENCE=(('A'..'Z', 'a'..'z'), ..
                        '0123456789', ('.', '!', '?'))
```

The defined collating sequence has four positions as follows:

1. All uppercase and lowercase letters.
2. All digits.
3. The characters ., !, and ?.
4. All other ASCII characters.

Examples

This directive specifies all default values for file $LOCAL.TEMP (abort at first error, include partition boundaries, copy only one record, use CYBER 180 format, the ASCII collating sequence, no exception records file, and no error recovery).

```
set_output_attributes, file=temp
```

This directive specifies attributes for the output file OUTFILE1:

```
set_output_attributes, ..
  file=outfile1, ..
  partition_disposition=exclude_boundaries, ..
  maximum_file_units=(100, records), ..
  condition_collating_sequence=lower_to_upper, ..
  error_disposition=no_abort, ..
  exception_records_file=invalid_records, ..
  conversion_error_disposition=recover
```

SET_SEQUENCE_ATTRIBUTES Directive

Purpose

Places a sequence number in each record of an output file. The sequence number replaces the contents of the designated field in the record.

Specify a separate SET_SEQUENCE_ATTRIBUTES directive for each output file you want to sequence. Each SET_SEQUENCE_ATTRIBUTES directive must follow the SET_OUTPUT_ATTRIBUTES directive specifying the name of the output file to be sequenced.

Format

SET_SEQUENCE_ATTRIBUTES (SET_SEQUENCE_ATTRIBUTE or
SETSA)
FILE=local_file_name
DUPLICATE_SPECIFICATION=local_file_name
SEQUENCE_FIELD=data_field_reference
SEQUENCE_NUMBER_PRESET=integer
SEQUENCE_NUMBER_INCREMENT=integer

Parameters

FILE(F)

Local file name of the output file in the working catalog. This parameter is required.

DUPLICATE_SPECIFICATION (DS)

Local file name whose SET_SEQUENCE_ATTRIBUTES directive specifications are to be duplicated for this file (see Duplicating Directive Specifications later in this chapter).

SEQUENCE_FIELD (SF)

Required field descriptor for the sequence field.

Field descriptor format: d[p,l]

d Data type (such as A for ASCII data).

p Beginning position of the sequence field in the output record. It can be specified as a single integer (the byte position) or as a set of two integers for data type B, that is, (byte,bit). It cannot be a function.

l Length of the sequence field. This must conform to rules of the selected data type.

The brackets are required; for more information, see the discussion of field descriptors in chapter 12.

SEQUENCE_NUMBER_PRESET (SNP)

Initial value (unsigned decimal integer) of the sequence number. The default is 1.

SEQUENCE_NUMBER_INCREMENT (SNI)

The sequence increment. This is an unsigned decimal integer. The default is 1.

Examples

The following directive sequence specify INFILE as the input file and OUTFILE1 as the output file. The SET_SEQUENCE_ATTRIBUTES directive specifies that a sequence number is to be entered in the 3-byte field beginning at byte 4 of each OUTFILE1 record.

```
SET_INPUT_ATTRIBUTES FILE=infile
SET_OUTPUT_ATTRIBUTES FILE=outfile1
SET_SEQUENCE_ATTRIBUTES FILE=OUTFILE1 ..
 SEQUENCE_FIELD=A[4,3]
```

The sequence field data type is ASCII (A). By default, sequence numbers start with 1 and are incremented by 1.

SET_PRINT_ATTRIBUTES Directive

Purpose Chooses formatting options for printing an output file.

Format

SET_PRINT_ATTRIBUTES (SET_PRINT_ATTRIBUTE or
SETPA)
FILE=local_file_name
DUPLICATE_SPECIFICATION=local_file_name
PRINT_FORMAT=keyword
PRINT_TITLE=literal

Parameters

FILE (F)

Local file name of the output file in the working catalog to be formatted for printing. The file organization of the file must be sequential. This parameter is required.

DUPLICATE_SPECIFICATION (DS)

Local file name whose SET_PRINT_ATTRIBUTES directive specifications are to be duplicated for this file (see Duplicating Directive Specifications later in this chapter).

PRINT_FORMAT (PF)

Print format. When the file attribute FILE_CONTENT is set to LIST, FMU inserts the necessary carriage control character:

Keyword	Meaning
1	Single space (default).
2	Double space.
3	Triple space.
DUMP	Dump option; output records are single spaced. A 30-byte record prefix, consisting of a decimal record number and character count, is printed for each record. The character count is the record length in characters that the record length would have been if SETPA had not been specified for the file.

PRINT_TITLE (PT)

Character string to be used as the print title. Must be a literal. The print title must not exceed 116 characters.

Remarks

- You must specify a separate SET_PRINT_ATTRIBUTES directive for each output file to be printed. A SET_OUTPUT_ATTRIBUTES directive for the file must precede the SET_PRINT_ATTRIBUTES directive.
- The print file can be a sequential or byte-addressable file; it cannot be a keyed file.
- The record type of the print file must be variable (V).
- The maximum record length of the print file must be at least the page width plus 1 (for the carriage control character).
- The print file is formatted so that a page number is printed on the first line of each page.
- If the FILE_CONTENT attribute value of the print file is LIST, an appropriate carriage control character is inserted in the first character position of each record. Otherwise, FMU inserts an appropriate number of blank lines to achieve the correct spacing.
- Your job must issue the command to print the file. (FMU can format a file for printing, but it cannot print it.)
- These file attributes are significant to SET_PRINT_ATTRIBUTES formatting:

  PAGE_FORMAT PAGE_WIDTH

  PAGE_LENGTH FILE_CONTENT

For more information, see the attribute descriptions in the SCL System Interface Usage manual.

Examples

The following directive sequence directs FMU to format each INFILE record for printing and then write it to OUTFILE1.

```
SET_INPUT_ATTRIBUTES  infile
SET_OUTPUT_ATTRIBUTES  outfile1
SET_PRINT_ATTRIBUTES  outfile1  PRINT_FORMAT=2,  ..
  PRINT_TITLE='OUTFILE1 CONTENTS'
```

The output file is formatted for double spacing and titling.

CREATE_OUTPUT_RECORD Directive

Purpose

Controls the order and format of data fields in each output record.

You can use the CREATE_OUTPUT_RECORD directive to:

- Rearrange data fields
- Insert literals at any position within a record
- Place the key of an input record into an output record
- Convert fields from one data type to another

Format

CREATE_OUTPUT_RECORD (CREOR)
FILE=local file name
DUPLICATE_SPECIFICATION=local file name
RECORD_PRESET_VALUE=keyword
Statement-List
CREATE_OUTPUT_RECORD_END (CREOREND)

Parameters

FILE (F)

Local file name of the output file in the working catalog. This parameter is required.

DUPLICATE_SPECIFICATION (DS)

Local file name whose CREATE_OUTPUT_RECORD directive specifications are to be duplicated for this file (see Duplicating Directive Specifications later in this chapter).

RECORD_PRESET_VALUE (RPV)

Value to which FMU sets fields not referenced in the CREOR statement-list.

NO_PRESET (NP)	No preset value (default)
CHARACTER_BLANK (CB)	Blank characters
CHARACTER_ZERO (CZ)	Zero characters

BINARY_ZERO (BZ)	Zero bits
INPUT_RECORD (IR)	Data in the output record is to be the same as the corresponding data in the input record, unless altered by assignment statements.

statement-list

(The statement list is not a parameter although it is part of the CREATE_OUTPUT_RECORD directive.)

Statements that perform data manipulations (see the statement descriptions in chapter 12.

Statement-list must be separated from the parameters by a statement separator (end-of-line or semicolon).

CREATE_OUTPUT_RECORD_END (CREOREND)

Required terminator for the CREOR directive. CREOR is the only directive with a terminator. You must use this terminator whenever you specify a CREOR directive.

Remarks

- Nonsignificant blanks can be included anywhere in the CREOR directive specification except within names, keywords, literals, ellipses, and relational operators.
- The CREATE_OUTPUT_RECORD directive formats the record according to your specifications. If you specify overlapping fields, your last specification establishes the final format.
- You must specify a separate CREATE_OUTPUT_RECORD directive for each output file to be reformatted. Only one CREATE_OUTPUT_RECORD can be used for an output file. You can duplicate the functions of one CREATE_OUTPUT_RECORD directive by specifying the DUPLICATE_SPECIFICATION parameter in the CREATE_OUTPUT_RECORD directive of another file (see the discussion under Duplicate_Specification Parameter in this chapter).

- The statement-list part of the directive specifies statements that perform the desired data manipulations. Statements consist of instructions that reformat input fields, test for conditions in the input record data, perform iterations on statements, and stop statement processing. See chapter 12 for detailed statement descriptions.

Examples

This directive sequence specifies the input file, an output file, and the data reformatting FMU is to perform on the records to be written to the output file.

```
SET_INPUT_ATTRIBUTES  infile
SET_OUTPUT_ATTRIBUTES  outfile ..
 ERROR_DISPOSITION=NO_ABORT
CREATE_OUTPUT_RECORD, FILE=outfile, ..
  RECORD_PRESET_VALUE=INPUT_RECORD
   IF A[2,3]='ABC' THEN
         A[2,3]='OUT'
   IFEND
CREATE_OUTPUT_RECORD END
```

Because the CREATE_OUTPUT_RECORD directive specified RECORD_PRESET_VALUE=INPUT_RECORD, the input fields that are not manipulated stay the same when output.

The IF statement can begin on the same line as the CREATE_OUTPUT_RECORD directive if it is separated from the parameter-list by a semicolon.

CREATE_OUTPUT_RECORD Statements 12

CREATE_OUTPUT_RECORD Statements 12

The statements described in this chapter are used in the statement-list part of the CREATE_OUTPUT_RECORD directive. You use them to specify exactly how you want your output record to be formatted.

Statement Conventions

This section gives the general rules for specifying statements in a CREATE_OUTPUT_RECORD statement list.

Statement Separation, Termination, and Continuation

The semicolon (;) is a statement separator and terminator. Multiple statement separators are considered to be one separator.

Statements can be written as follows:

```
statement [; statement] ...

or

statement
[statement]
   :
```

where the end of a line is also a statement separator and terminator.

Statement continuation is done by placing two or more periods at the end of a statement line.

Comment Insertion

You can insert comments anywhere that blanks can be inserted. They are processed as blanks.

The quotation mark (") signifies the start of a comment. A comment can be ended by either a closing quotation mark or the end of a line.

You can continue a comment with two or more periods.

Structured Statements

A structured statement is a statement that contains a statement list. The structured statements are BLOCK, FOR, IF, LOOP, REPEAT, and WHILE. In general, the FMU structured statements are used like their SCL counterparts.

A structured statement can be labeled. The label allows an EXIT or CYCLE statement to reference the structured statement. The EXIT or CYCLE statement must be in the statement list of the structured statement.

The label can be any valid SCL name. A preceding label must end with a colon (:). No space can precede the colon; spaces can follow the colon.

Logical Expressions

A logical expression is an expression that can be evaluated as true or false. Logical expressions are required on IF, REPEAT, and WHILE statements and are optional on CYCLE and EXIT statements.

A logical expression can be a single boolean expression or it can combine two or more boolean expressions with logical operators. You can negate any boolean expression by preceding it with the word NOT.

One of the following logical operators combines each pair of boolean expressions in a logical expression:

AND Both boolean expressions must be true for the logical expression to be true.

OR One of the boolean expressions must be true for the logical expression to be true.

XOR One, but not the other, of the boolean expressions must be true for the logical expression to be true.

AND is evaluated before OR or XOR; OR and XOR have the same precedence.

A boolean expression is one of the following:

- Field descriptors of data type L (the contents of the input field is interpreted as a true or false value).

- Intrinsic functions that return a boolean value ($IN_RECORD or $VALID_DATA)For more information, see the function descriptions in chapter 13.
- Relational expressions.

A relational expression compares two items using one of these operators:

<	Less than	>	Greater than
<=	Less than or equal to	>=	Greater than or equal to
=	Equal to	<>	Not equal to

A relational expression can compare any two of these items:

- Intrinsic functions (valid only to the left of the operator)
- Contents of an input record field (specified as a field descriptor or the word KEY)
- Literal enclosed in apostrophes (' ')
- Arithmetic expression (for more information, see the discussion of arithmetic expressions in chapter 13).

Assignment Statement

An assignment statement causes a value to be stored in the output record. The value can be a literal or the value of a data field in the input record. If an input record data field is specified, the assignment statement also specifies the data reformatting performed on the field.

An assignment statement uses one of the following formats:

destination-item = source-item

or

source-item

A source item can be a field descriptor, a literal, or the keyword KEY. (Field descriptors are described in chapter 13.) KEY, on the right side of the equal sign, specifies that the data to be moved or reformatted is the primary-key value of the input record. The key of the input record is the data in the primary-key field defined by the KEY_POSITION and KEY_LENGTH attributes of the input file.

A destination item can be a field descriptor or the keyword KEY. (Field descriptors are described in chapter 13.) KEY, on the left side of the equal sign, specifies that the data is to be stored in the key field of the output record. The output key field is defined by the KEY_LENGTH and KEY_POSITION attributes of the output file.

These are some examples of assignment statements:

`A[3,8] = A[7,8]` Moves the 8-byte alphanumeric field beginning at byte 7 in the input record to the 8-byte alphanumeric field beginning at byte 3 in the output record. Both source and destination fields are described by field descriptors.

`A[4,3] = 'ABC'` Moves the literal ABC to the alphanumeric (A) output field beginning at byte 4; the output field is three bytes long.

`A[1,9] = Z[1,10]` Moves Z-type data in the first 10 bytes of the input record to the first nine bytes of the output record. Data is converted from integer (Z) to alphanumeric (A) and truncated. (For information on data type conversions, see appendix F.)

`N[1,5] = KEY`	Moves the key in the input record to the 5-byte normal integer (N) field at the beginning of the output record. The key field is defined by the KEY_POSITION and KEY_LENGTH attributes of the input file.
`KEY = N[1,7]`	Makes the 7-byte normal integer (N) field of the input record the key of the output record. The key field is defined by the KEY_POSITION and KEY_LENGTH attributes of the output file.
`KEY = KEY`	Moves the input key value to the output key field. The keys are defined by the KEY_POSITION and KEY_LENGTH attributes of the input and output files.

Reformatting Considerations

This section discusses the data field reformatting considerations when using the assignment statement.

Field Length

The source-item value is reformatted to fit the length of the destination-item field. A character string that is too long is truncated on the right. A character string that is too short is left-justified and blank-filled.

If no length is specified for the destination item, the default length for the destination-item data type is used.

Specifying zero as the length of a source data field has special significance. A zero-length A field represents a blank. A zero-length L field represents FALSE. A zero length for all the other data types is taken as zero.

If you specify zero as the length on the destination field descriptor, the source field is skipped. Specifying a zero-length destination field has the effect of establishing the current output position (the value that would be returned by the $CURRENT_OUTPUT_POS intrinsic function). You can use the zero-length destination item, together with the RECORD_PRESET_VALUE parameter, to add padding characters at the end of the output record.

If a zero-length record results from statement-list operations, a zero-length record is written to the output file. For example, assuming that the RECORD_PRESET_VALUE parameter is not set to INPUT_RECORD, these statements write a zero-length record if the test for A[1,1] = A[3,1] is false.

```
CREATE_OUTPUT_RECORD, FILE=OUTFILE1, ..
 RECORD_PRESET_VALUE=NO_PRESET
    IF A[1,1] = A[3,1] THEN
        A[2,1] = A[3,1]
    IFEND
CREATE_OUTPUT_RECORD_END
```

Current Position Pointers

FMU maintains pointers for the input and output records. Each pointer initially points to byte 1, bit 1 of the record. After an assignment statement is executed, the pointers are reset to point to the byte (or bit for data type B) following the field just processed.

If you omit a position in a field descriptor, the pointer value is used as the default value for the position. The default position for a field descriptor on the left of the equal sign in an assignment statement is the value of the output record pointer. The default position for a field descriptor on the right of the equal position of the input record is accessed. If the data type is not B and the current bit index is greater than 1, the pointer is advanced to bit 1 of the next byte position.

To specify the input and output position pointer values explicitly in a field descriptor, use the intrinsic functions $CURRENT_INPUT_POS and $CURRENT_OUTPUT_POS. (Intrinsic functions are described in chapter 13.)

Single Field Descriptor

When an assignment statement consists of a single field descriptor, the field descriptor describes the source item. The destination item is a field having the same data type and length as the source item and beginning at the current output position pointer in the output record.

If length is not specified in the single field descriptor, the default lengths, based on data types and the MACHINE_FORMAT (MF) parameter value, are used for source and destination fields. (Data types are described in chapter 13.)

The following examples show assignment statements using single field descriptors given with their equivalent assignment statements:

`A[4,3]` is equivalent to `A[ ,3]=A[4,3]` or `A[$COP, 3]=A[4,3]`.

`A[ ,3]` is equivalent to `A[ ,3]=A[ ,3]` or `A[$COP, 3]=A[$CIP, 3]`.

`A` is equivalent to `A=A` or `A[,1]=A[,1]` or `A[$COP,1]=A[$CIP,1]`.

The single field descriptor is useful when you reference variable-length fields, because the destination field equals the length of the source field. An example of this is when the position and length of a source field are defined record by record by the $INPUT_STRING_POS function.

Data Alignment

Alignment is never forced to a word boundary. If you need to align data by word boundary or by other means, you must supply the proper fill items explicitly, or specify position.

Block/Blockend Statement

Purpose Groups a sequence of statements into a block.

Format

label: **BLOCK**
statement_list
BLOCKEND *label*

Variable Elements

label

Optional label for statement. The label can be referenced by EXIT statements in the statement list.

The label is optional. You can specify a preceding label without a trailing label. If you specify both labels, they must be identical.

statement_list

Sequence of one or more statements. The statement list is executed once unless an EXIT or STOP statement ends its execution.

Examples The following BLOCK statement has a statement list of three statements. The first assignment is always done; the second assignment is done only if the condition on the EXIT statement is false.

```
prefix: BLOCK
          A[1,5] = I[1,8]
          EXIT prefix WHEN $INPUT_RECORD_LENGTH = 8
          A[6,5] = I[1,8]
        BLOCKEND prefix
```

Cycle Statement

Purpose Initiates immediate execution of the next iteration of an enclosing statement list. It can be used in LOOP, FOR, WHILE, and REPEAT statements.

Format

CYCLE *label*
WHEN logical_expression

Variable Elements

label

Optional label specifying the enclosing statement to be cycled. If the label is omitted, the innermost LOOP, FOR, WHILE, or REPEAT statement is cycled.

WHEN logical_expression

Optional condition tested to determine if the statement is cycled. If the condition is evaluated TRUE, the statement is cycled; if the condition is evaluated FALSE, the statement is not cycled and processing of the statement list continues.

Remarks

- The following lists the processing performed when the statement is cycled:

LOOP	The first statement in the LOOP statement list is processed.
FOR	The iteration count is incremented and compared with the final value.
WHILE or REPEAT	The condition controlling iteration of the statement is evaluated to determine if the statement list is repeated.

- FMU diagnoses these CYCLE errors:
  - The specified label does not match the label of an enclosing FOR, LOOP, REPEAT, or WHILE statement.
  - The CYCLE statement has no label and it is not in a FOR, LOOP, REPEAT, or WHILE statement list.

Examples This statement cycles the FOR statement when the IF statement condition is true.

```
FOR iterations = 1 TO 2 DO
  "assignment statement"
  IF I[$CURRENT_INPUT_POS,8] > 99999
    CYCLE
  IFEND
  "assignment statement"
FOREND
```

Exit Statement

Purpose Ends execution of the statement list in which it occurs.

Format

EXIT *label*
WHEN logical_expression

Variable Elements:

label

Optional label specifying the enclosing statement to be ended. If the label is omitted, the innermost BLOCK, FOR, IF, LOOP, REPEAT, or WHILE statement is ended.

WHEN logical_expression

Optional condition tested to determine if the statement is ended. If the condition is evaluated TRUE, the statement is ended. If the condition is evaluated FALSE, the statement is not ended; processing of the statement continues.

The word WHEN can be omitted.

Examples The following statements show two nested blocks (the second block is enclosed by the first block). The first EXIT statement ends the first block. The second EXIT statement ends the second block.

```
first: BLOCK
  second: BLOCK
            IF A[1,1] = A[2,1]
              EXIT first
            ELSEIF A[1,1] = A[3,1]
              EXIT second
            IFEND
            "assignment statement"
          BLOCKEND second
          "assignment statement"
        BLOCKEND first
```

For/Forend Statement

Purpose Unconditionally repeats execution of a statement list a number of times.

Format

label: **FOR name = initial TO final** *BY step DO;*
statement_list
FOREND *label*

Variable Elements:

label

Optional label for statement. The label can be referenced by CYCLE and EXIT statements in the statement list.

The label is optional. You can specify a preceding label without a trailing label. If you specify both labels, they must be identical.

name

SCL name for the loop index variable. It should not be an FMU reserved word. The reserved words are listed in the following table.

In a nested FOR loop, the inner FOR statements must not use the same name as the outer FOR statements.

initial

Initial value of the iteration count. This value is an integer or arithmetic expression.

final

Maximum value of the iteration count. It is an integer, an arithmetic expression, or a field descriptor. When a field decriptor is specified, the value in the field is converted to an integer value if necessary.

BY step

Increment value specified as an integer constant or arithmetic expression. Unlike other integer values, it can be a negative value, causing the index to decrement instead of increment. If it is omitted, the iteration count is incremented by 1.

You can omit the word BY.

statement_list

Sequence of one or more statements executed once for each value of the index count.

Reserved Words

A	ELSEIF	KEY	STOP
AND	EXIT	L	THEN
B	F	LOOP	TO
BY	FOR	LOOPEND	U
BLOCK	FOREND	MOD	UNTIL
BLOCKEND	G	N	WHEN
CREOR	H	NOT	WHILE
CREOREND	I	OR	WHILEND
CYCLE	IF	P	XOR
DO	IFEND	Q	Y
ELSE	J	REPEAT	Z

Remarks

- The FOR phrase must end with the word DO, a semicolon and/or the end of the line.
- The index variable is set to the initial value when FOR statement processing begins. The index variable can be referenced within the FOR statement. It is incremented by the step when a CYCLE statement is processed and at the end of each iteration of the statement list.
- The incremented index value is then compared with the final statement list and is executed again. If the index value is greater than or equal to the final value processing continues with the statement following the FOREND.
- The current value of the loop index can be referenced by the name, but a new value cannot be assigned to the name.

Examples

This FOR statement moves three contiguous 3-byte fields in the input record to three contiguous 4-byte fields in the output record, converting the data from type A to type Z.

```
FOR DUMMY = 1 to 3 DO
  Z[ ,4] = A[ ,3]
FOREND
```

The following shows the data from an input record moved to the output record:

Input record: 123456789

Output record: +123+456+789

This FOR statement iterates three times. (The FOR range is from 2 to 6 incremented by 2.) At each iteration, a search is made of the input record for the string k (lowercase) beginning at the current pointer position. As each k is found, it is placed in the output record at the same position as in the input record.

```
FOR dumname = 2 TO 6 BY 2 DO
  A[$INPUT_FIELD_POS,1] = A[$INPUT_STRING_POS('k',..
    $CURRENT_INPUT_POS,1,FIRST_CHARACTER),1]
FOREND
```

Input record: aaakaaakaaak

Output record: k k k

(The preceding example assumes that RECORD_PRESET_VALUE=CHARACTER_BLANK is specified on the CREATE_OUTPUT_RECORD directive.)

The number of times the following FOR statement iterates is equivalent to the integer value in the I[10,2] field of the input record. Each iteration moves a three-character field from the current position of the input record to the current position of the output record.

```
FOR I = 1 TO I[10,2] DO
  A[,3] = A[,3]
FOREND
```

If/Elseif/Else/Ifend Statement

Purpose Executes one of the statement lists in the statement if the condition for the statement list is evaluated as true.

Format

IF logical_expression *THEN;*
statement_list
ELSEIF logical_expression;
statement_list
ELSE; statement_list
IFEND

Variable Elements:

logical_expression

Expression evaluated to determine if the following statement list is executed. If the expression is evaluated as true, the statement list is executed. If the expression is evaluated as false, the statement list is not executed and processing continues with the next part of the IF statement.

statement_list

Sequence of one or more statements executed if the preceding logical_expression is evaluated as true or, for the ELSE statement list, if all preceding conditions are evaluated as false.

Remarks

- Only the IF phrase, its statement list, and the IFEND terminator are required. The ELSEIF and ELSE phrases and their statement lists are optional. You can specify multiple ELSEIF phrases and statement lists, but only one ELSE phrase and statement list.
- The IF phrase must end with the word THEN, a semicolon (;) or the end of the line. Each ELSEIF and ELSE phrase must end with semicolon (;) or the end of the line.

Examples The following IF statement list contains a nested IF statement.

```
"1"  IF Z[1,3] = '+01'          "If true, do 2;
"2"    A[5,2] = A[5,2]          "if false, do 9.
"3"    IF A[14,1] > A[15,1]     "If true, do 4;
"4"      Z[14,2] = '+1'         "if false, do 6.
"5"    ELSE                     "This ELSE belongs to
"6"      Z[14,2] = '+2'         "the IF at 3.
"7"    IFEND
"8"
"9"  ELSEIF Z[1,3] = '+02'      "If true, do 10;
"10"   A[5,2] = A[9,2]          "if false, do 12.
"11"
"12" ELSEIF Z[1,3] = '=03'      "If true, do 13;
"13"   A[5,2] = A[11,1]         "if false, do 14.
"14" IFEND
```

Loop/Loopend Statement

Purpose Repeats a statement list an unlimited number of times.

Format

label: **LOOP**
statement_list
LOOPEND *label*

Variable Elements:

label

Optional label for statement. The label can be referenced by CYCLE and EXIT statements in the statement list.

The label is optional. You can specify a preceding label without a trailing label. If you specify both labels, they must be identical.

statement_list

Sequence of one or more statements. The statement list is executed repeatedly until an EXIT or STOP statement ends its execution.

Examples The following LOOP statement repeats until the IF condition is evaluated as true and the STOP statement is executed.

```
LOOP
  "assignment statement"
  IF $CURRENT_INPUT_POS = $INPUT_RECORD_LENGTH
    STOP
  IFEND
  "assignment statement"
LOOPEND
```

Repeat/Until Statement

Purpose Repeats a statement list until a condition is true. The statement list is executed at least once.

Format

label: **REPEAT**
statement_list
UNTIL logical_expression

Variable Elements:

label

Optional label for statement. The label can be referenced by CYCLE and EXIT statements in the statement list.

The label is optional. Unlike other statements, a trailing label is not valid on a REPEAT statement.

statement_list

Sequence of one or more statements. The statement list is executed repeatedly until the logical_expression is true or an EXIT or STOP statement ends its execution.

logical_expression

Required expression evaluated after each repetition of the statement list. If the expression is evaluated true, the statement following the UNTIL is executed. If the expression is evaluated false, the REPEAT statement_list is executed again.

Examples The following REPEAT statement repeats until either the EXIT condition or the UNTIL condition is true.

```
REPEAT
  "assignment statement"
  EXIT WHEN A[1,9] = 'BLUE MOON'
  "assignment statement"
UNTIL $CURRENT_OUTPUT_POS = 80
```

Stop Statement

Purpose Immediately ends formatting of the current output record by the CREATE_OUTPUT_RECORD statement list. Any unassigned fields in the record are assigned the value specified by the RECORD_PRESET_VALUE parameter on the CREATE_OUTPUT_RECORD directive.

Format **STOP**

Examples The following CREATE_OUTPUT_RECORD statement list contains a STOP.

```
CREATE_OUTPUT_RECORD, FILE=outfile ..
 RECORD_PRESET_VALUE= INPUT_RECORD
   A[1,2] = '01'
   A[3,4] = '0002'
   IF N[6,10] > '9999'
      STOP
   IFEND
   IF A[15,,17] = A[20,,22]
      A[15,,17] = A[15,,17]
   ELSE
      A[15,,17] = 'EEE'
   IFEND
CREOREND
```

The field described by Z[6,10] is the record number. Additional output formatting is to be done for record numbers less than or equal to 9999. Once that number has been surpassed, only the literals are to be placed in the next record.

When 9999 is passed, the STOP statement stops statement list processing on the record. Because the RECORD_PRESET_VALUE is INPUT_RECORD, the unassigned output fields are assigned the contents of the corresponding fields of the input record. Processing continues with the next record.

While/Whilend Statement

Purpose Repeats a statement list while a condition is true. The statement list is not executed if the condition is initially false.

Format

label: **WHILE logical_expression DO**
statement_list
WHILEND *label*

Variable Elements:

label

Optional label for statement. The label can be referenced by CYCLE and EXIT statements in the statement list.

The label is optional. You can specify a preceding label without a trailing label. If you specify both labels, they must be identical.

logical_expression

Required expression evaluated before each repetition of the statement list. If it is evaluated as true, the statement list is executed. If it is evaluated as false, execution continues with the statement following the WHILEND.

statement_list

Sequence of one or more statements. The statement list is executed repeatedly while the logical_expression is true or until an EXIT or STOP statement ends its execution.

Examples The following WHILE statement repeats until either the WHILE condition or the EXIT condition is true.

```
WHILE $CURRENT_OUTPUT_POS <= 80 DO
    "assignment statement"
    EXIT WHEN A[1,9] = 'FULL MOON'
    "assignment statement"
WHILEND
```

Data Field Referencing 13

Data Field Referencing 13

This chapter describes the means of specifying FMU data fields. Data field references are used in the CREATE_OUTPUT_RECORD statements described in chapter 11 and to specify the sequence field on the SET_SEQUENCE_ATTRIBUTES directive described in chapter 11.

Field Descriptors

A field descriptor describes a field in an input or output record. To describe a field, a field descriptor specifies the data type and the position and length of the field.

The data type specifies the representation of the data in the field. For example, data type A specifies an ASCII string.

The field starting position is specified as the first byte or bit in the field. You can specify the length as either the number of bytes or bits in the field or the position of the last byte or bit in the field.

Field Descriptor Format

A field descriptor can have one of the following formats. The brackets ([ ]) are part of the formats. Everything between the brackets is optional. If any of the items within the brackets are used, the brackets are required.

data-type

data-type[start-position,length]

data-type[start-position,,trailing-position]

(Notice the two commas in this format.)

data-type

The one-character mnemonic for an FMU data type. The data types are described later in this chapter.

start-position

Starting position of the field. For data type B, it is a bit position. For all other data types, it is a byte position.

A byte position is specified as a single value; a bit position is specified as two values as follows:

(byte,bit)

A byte value or a bit value can be specified as an unsigned integer, a position intrinsic function, or a field descriptor. Bytes and bits are numbered from the left beginning with 1.

Default start-position:

The default byte position is the current byte position (the value that would be returned at this point by a $CURRENT_INPUT_POSITION or $CURRENT_OUTPUT_POSITION function). The value returned is the byte position immediately following the last field referenced. (If no field was previously referenced, start-position 1 is used.)

When specifying a field descriptor for data type B, the default bit position depends on how the byte position is specified:

- If the byte position is omitted, the default is the current bit position (the value that would be returned at this point by a $CURRENT_INPUT_BIT_POS or $CURRENT_OUTPUT_BIT_POS function).
- If the byte position is specified as an integer or field descriptor, the default bit position is 1.
- If the byte position is specified by an intrinsic function, the default bit position is the default value for the corresponding bit-position intrinsic function. For example, the bit-position function corresponding to the byte-position function $INPUT_FIELD_POS is $INPUT_FIELD_BIT_POS.

length

Field length in bytes (or in bits for a data type B). It can be specified as an integer constant, a length intrinsic function, or a field descriptor.

The default value is the default length for the data type. To see the default lengths for NOS/VE data types, see Data Types in this chapter.

trailing-position

Designates the rightmost, or ending, position of the field.

This is an alternate specification for length. You can specify either length or trailing-position in a field descriptor but not both. Trailing-position can be specified in the same way as start-position.

Field Descriptor Examples

The following are examples of field descriptors:

`A[5,10]`	Alphanumeric data type, beginning at byte position 5, and 10 bytes long.
`A[5,,14]`	Alphanumeric data type, beginning at at byte position 5, ending with byte position 14. Equivalent to the previous example.
`A`	Alphanumeric data type, equivalent to A[,1], beginning at the current pointer position with a length of 1 byte.
`Z[2,10]`	Type Z (integer character string with leading zeros and leading sign), beginning at byte position 2 with a length of 10 bytes.
`B[(5,2),8]`	Type B, beginning at bit number 2 of byte position 5 with a length of 8 bits.
`B[(5,2),,(8,3)]`	Type B, beginning at bit number 2 of byte position 5 and ending on bit number 3 of byte position 8.
`A[$CIP,4]`	Type A, beginning at the current input byte position as returned by the intrinsic function $CURRENT_INPUT_POS ($CIP) with a length of 4 bytes. (The $CIP intrinsic function is described later in this chapter.)

`A[$CIP+1,4]`	Type A, beginning at the next byte after the current input byte position as found by the intrinsic function $CIP with a length of 4 bytes.
`A[3,I[1,2]]`	Type A, beginning at byte 3 and ending at the byte position specified by the value in the I[1,2] field. This is an example of a nested field descriptor as described in the next section.

Nested Field Descriptors

The position or length of a field can be specified by the contents of another field in the record. This is specified using a nested field descriptor.

For example, a variable-length string starts at byte 6 and has a 2-byte integer header (beginning at byte 4) that specifies the string length. The field descriptor for the variable-length string field is as follows:

```
A [6,I [4,2] ]
```

To interpret a field descriptor containing a nested field descriptor, FMU first gets the value of the field described by the nested field descriptor.

If necessary, FMU converts the value of the nested field into a 64-bit integer using the standard A-to-I transformation. (The A-to-I transformation rules are listed in appendix F.)

If an error occurs during conversion of a nested field value, the output record being constructed is discarded. (FMU does not assign a default value to a nested field.) Further processing depends on the specified ERROR_DISPOSITION parameter value (ABORT or NO_ABORT).

Data Types

These are the valid FMU data types:

Character Strings:

Data Type	Description	Maximum Length	Default Length
A	Any ASCII characters	None	1
G	Characters representing a floating point number	40	22

Character Representations of Integers:

Data Type	Description	Maximum Length	Default Length
N	Leading spaces, floating negative sign	None	19
Y	Leading zeros, sign is rightmost character	38	19
Z	Leading zeros, sign is leftmost character	38	19
H	Trailing combined sign	38	18
J	Leading combined sign	38	18

Numeric Arithmetic Representations:

Data Type	Description	Maximum Length	Default Length
B	Unsigned binary	128	1
F	Floating point	8 or 16	8
I	Signed integer	8	8
L	Logical	8	8
P	Signed packed decimal	19	10
Q	Unsigned packed decimal	19	10
U	Unsigned unpacked decimal	38	18

Data type I is a signed integer. The leftmost bit is always considered a sign bit. For an unsigned integer, use data type B.

NOS/VE FORTRAN Data Types

If the data was written by a NOS/VE FORTRAN program, use the data type corresponding to the FORTRAN data declaration as follows:

Data Type	NOS/VE FORTRAN	Data Type	NOS/VE FORTRAN
A	CHARACTER	I	INTEGER
B	Any	J	CHARACTER
F [,8]	REAL	L	LOGICAL
F [,16]	DOUBLE PRECISION	N	CHARACTER
G	CHARACTER	Y	CHARACTER
H	CHARACTER	Z	CHARACTER

The notation F [,8] refers to a an 8-byte NOS/VE floating point number. F [,16] refers to a 16-byte NOS/VE floating point number. FORTRAN type COMPLEX data is handled as two fields of type F.

NOS/VE COBOL Data Types

If the data was written by a NOS/VE COBOL program, use the data type corresponding to the COBOL data declaration as follows:

Data Type	NOS/VE COBOL
B	Any
F [,10/8]	COMP-1
F [,20/16]	COMP-2
I	COMP PIC S9...
P	COMP-3 PIC S9...
Q	COMP-3 PIC 99...
U	PIC 99...
A	PIC X
H	Any PIC S9... SIGN IS TRAILING
J	Any PIC S9... SIGN IS LEADING
N	Any PIC ...ZZZ9
Y	Any PIC S9... SIGN IS TRAILING SEPARATE
Z	Any PIC S9... SIGN IS LEADING SEPARATE

For COMP PIC S9... data, COBOL determines the number of bytes from the number of nines in the PICTURE clause.

Maximum Precision of NOS/VE Data Types

The following table lists the maximum data precision possible for each FMU data type.

Data Type	Precision in Digits (Rounded Up to the Nearest Integer)	Number of Guaranteed Accurate Decimal Digits (Rounded Down to the Nearest Integer)
F[,8]	15	14
F[,16]	29	28
G[,m]	m	m
H[,m]	m	m
I[,m]	$\log_{10}(2^{-(8*m)-1}1)$	$\log_{10}(2^{-(8*m)-1}1)$
J[,m]	m	m
N[,m]	min(m, 19)	min(m, 18)
P[,m]	?m-1	2m-1
Q[,m]	2m	2m
U[,m]	m	m
Y[,m]	m-1	m-1
Z[,m]	m-1	m-1

Data Type A

Data type A is an alphanumeric character string. It can include any member of the ASCII character set. The length is not restricted to 256 bytes, nor is there any restriction as to field contents.

For an A to A conversion, a left-to-right byte move is performed until the destination field is full. If the source is prematurely exhausted, the remainder of the destination is blank-filled.

Examples `A[1,10] = 'alpha23` Output: `|a|l|p|h|a|2|3| | | |`

Data Type B

Data type B is an unsigned binary integer. The field does not have to begin on a byte boundary (that is, be byte-aligned). The length is restricted to 128 bits on NOS/VE. An attempt to assign a negative value to an unsigned binary field results in an error.

Examples

`B[1,, (2,3)] = '10011101001(2)'`	The base 2 number is placed in an 11-bit field beginning at bit 1 of byte 1.
`B[1,8] = '11110001(2)'`	Eight binary digits are placed in an 8-bit field beginning at bit 1 of byte 1.
`B[(5,2),8]='10101111(2)'`	The binary digits are stored in an 8-bit field beginning at bit 2 of byte 5.
`B[1,7] = '123'`	The binary form of the literal (decimal number 123) is placed in a 7-bit field beginning at bit 1 of byte 1.

Data Type F

Data type F is floating-point data. It is unlike the other data types in that only two lengths are permissible: 8 bytes and 16 bytes. F(8) represents single-precision (FORTRAN REAL) data and F(16) represents double-precision floating-point data. The minimum (and maximum) length is one word for single precision and two words for double precision.

Examples These values are stored in eight bytes each:

```
F[2,8]='1234'
F[4,8]='12345678901234'
F[3,8]='1.23456789E+13'
```

This value is stored in 16 bytes:

```
F[5,16]='1.2345678901234567890 12'
```

Data Type G

Data type G is the general character representation for floating-point numbers. The field length is limited to 40 bytes. When the exponent of the number to be output is in the range:

```
-6 <= x < 9
```

the number is output in F style. Otherwise, the number is output in E style.

Examples `G[1,10]='12345678901234'` Output: `|1|.|2|3|4|5|D|+|1|3|`

`G[1,5] = '25'` Output: `| | |2|5|.|`

For more information on the E format descriptor, see the FORTRAN Language Definition Usage manual.

Data Type H

Data type H is for trailing-sign Hollerith data. Data type H offers the most precision of all the data types, with up to 38 decimal digits.

The sign of the field and its low-order (units) numeric digit are contained in the low-order byte as shown in table 13-1.

In COBOL, this data is described by PICTURE IS S9(n) SIGN IS TRAILING, USAGE IS DISPLAY.

Examples `H[1,4] = '50'` Output: `|0|0|5|{|`

Table 13-1. Sign Position for H Fields

Low-Order Digit	Character Placed in That Position + (Sign)	Character Placed in That Position - (Sign)
0	{	}
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

Data Type I

Data type I is a signed integer. It is a variable-length binary format, with the leftmost bit being the sign bit.

NOS/VE data in this format can vary in length from 1 to 8 bytes, where a byte consists of 8 bits. Negative values are represented in the two's complement form.

Examples `I[3,4]= '25'` The source is converted to a binary number located in bytes 3 through 6.

Data Type J

Data type J is for leading-sign Hollerith data. The sign of the field and high-order numeric digit are contained in the high-order byte as shown in the table 13-2.

In COBOL, this data is described by PICTURE IS S9(n) SIGN IS LEADING, USAGE IS DISPLAY.

Examples `J[1,4] = '50'` Output: `|0|0|E|0|`

Table 13-2. Sign Position for H Fields

Low-Order Digit	Character Placed in That Position + (Sign)	Character Placed in That Position - (Sign)
0	{	}
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

Data Type L

Data type L is the FMU equivalent of FORTRAN LOGICAL. If the sign bit (leftmost bit) of the source field is zero, the value is taken as FALSE. Otherwise, the value of the source field is taken as TRUE. For NOS/VE data, the length can be from 1 through 8 bytes.

For a transformation of data from character data to logical (L), see Data Type Conversion Between NOS/VE Files in appendix F.

Examples

```
L[3,8]='.TRUE.'
```

The output field is binary data with a nonzero sign bit.

```
L[3,4]=A[5,4]
```

The sign bit of the destination is zero or nonzero (FALSE or TRUE) depending on the source data.

Data Type N

Data type N is the integer character string with leading blanks. There is no length restriction for this decimal integer format. The value representable, however, is limited to a value that is representable by data type I.

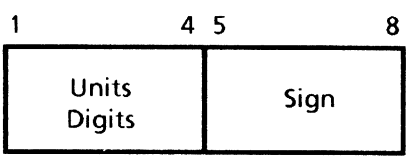
The character digits are right-justified in the field, with blank fill. If the number is negative, a negative sign replaces the rightmost blank. The contained value must be representable in a 64-bit two's complement word.

Examples

```
N[4,8] = '325'      Output: | | | | |3|2|5|
N[2,4] = '-12'      Output: | |-|1|2|
N[1,3] = '+123'     Output: | | |1|2|3|
```

Data Type P

Data type P is for signed packed-decimal data. Two decimal binary digits are packed per byte, up to a total of 37 digits. The sign of the field and the low-order, units digit are contained in the low-order (rightmost) byte, as follows:



In COBOL, this data type can be described by PICTURE IS S9(n) USAGE IS PACKED-DECIMAL.

Examples

```
P[4,2]= '12'        Output: |_|+|1|2|
```

Each digit is a binary digit contained in a half byte. Valid sign codes are as follows:

Hex	Bit Pattern	Sign	
A	1010	+	
B	1011	+	
C	1100	+	Preferred code used when a P field is generated.
D	1101	-	Preferred code used when a P field is generated.
E	1110	+	
F		+	

Data Type Q

Data type Q is the unsigned packed decimal data type. Two decimal binary digits are packed per byte, up to a total of 38 digits (19 bytes).

The data field is always an integral number of bytes. If the number has an odd number of digits, the leftmost 4 bits of the leftmost bytes are zero.

In COBOL, this data type can be described by:

PICTURE IS 9(n) USAGE IS PACKED-DECIMAL.

Examples `Q[4,2]= '12'` Output: `0|0|1|2|`

Data Type U

Data type U is the unsigned unpacked decimal data type. Each byte contains one right-justified decimal digit with zero fill up to 38 digits (19 bytes).

In COBOL, this data type can be described by:

PICTURE IS 9(n).

Examples `U[4,2]= '12'` Output: `|0|1|0|2|`

Data Type Y

Data type Y is an integer character string, an alphanumeric representation of up to 37 decimal integer digits with trailing sign. The field is zero-filled to the left of the most significant digit. Its minimum length is two and includes the sign.

In COBOL, this data type is described by PICTURE IS S9(n) SIGN IS TRAILING SEPARATE, USAGE IS DISPLAY.

Examples `Y[3, 5] = '25'` Output: `|0|0|2|5|+|`

Data Type Z

Data type Z is an integer character string, an alphanumeric representation of up to 37 decimal integer digits with leading sign, and character zero fill between the sign and the most significant digit. Its minimum length is two and includes the sign.

In COBOL, this data type is described by PICTURE IS S9(n) SIGN IS LEADING SEPARATE, USAGE IS DISPLAY.

Examples `Z[3, 5] = '25'` Output: `|+|0|0|2|5|`

Intrinsic Functions

You can write field descriptors with explicit values designating position and length. For example, the field descriptors N[1,4] and Z[3,,10] specify explicit values.

However, when you do not know the exact location or length of a data item, such as when data has variable length or position, you can use intrinsic functions to get the correct value.

The intrinsic functions can be categorized as follows:

- Position Functions
- Length Functions
- String Search Functions

Position Functions

Position functions are used in the start-position and trailing-position parts of a field descriptor to establish those positions. Position functions return either a byte position or bit position. Bit-position functions are used only for the bit-position portions of a field descriptor and only when the data type is B.

$CURRENT_INPUT_POS ($CIP)

The $CURRENT_INPUT_POS ($CIP) function returns the current input position. The current input position is the byte after the last byte that has been processed by a source item in a CREATE_OUTPUT_RECORD assignment statement.

For example, if the last byte processed in the source item A[2,4] is 5 and [$CIP,3] is in the next assignment statement, $CIP would yield byte position 6.

$CURRENT_INPUT_BIT_POS ($CIBP)

The $CURRENT_INPUT_BIT_POS ($CIBP) function returns the last input record bit position established by a source item of a CREATE_OUTPUT_RECORD assignment statement that has just been executed. The current input position is the bit after the last bit in the input record processed by a source item in an assignment statement.

For example, suppose the last bit processed in the source item B[(5,2),8] is bit 1 of byte 6. $CIBP at this time has a value of 2. The function is used in a subsequent source item B[(9,$CIBP),8].

$CURRENT_OUTPUT_POS ($COP)

The $CURRENT_OUTPUT_POS ($COP) returns the current output position.

The current output position in the output record is the byte after the last byte processed by a destination item in a CREATE_OUTPUT_RECORD assignment statement.

For example, the last byte processed by a destination item Z[8,,11] is 11. If [$COP] is in the next assignment statement, it returns byte position 12.

The following example demonstrates the use of $CIP and $COP:

```
A[1,1] = A[2,1]
A[$COP,1] = A[$CIP,1]     "$COP = 2, $CIP = 3"
A[$CIP,2] = A[2,2]        "$CIP = 4"
```

$CURRENT_OUTPUT_BIT_POS ($COBP)

The $CURRENT_OUTPUT_BIT POS ($COBP) function returns the last output record bit position established by a destination item of a CREATE_OUTPUT_RECORD assignment statement that has just been executed. Current output bit position is the bit in the output record after the last bit processed by a destination item in an assignment statement.

For example, the last bit processed in the destination item B[(3,1),4] is bit 4 of byte 3. $COBP at this time has the value of bit 5 and can be used in a subsequent destination item such as B[(,$COBP),4].

$INPUT_FIELD_POS ($IFP)

The $INPUT_FIELD_POS ($IFP) returns the position part of the last established source item. $IFP includes only the byte position of the last source item referenced.

For example, if the last established source item in the assignment statement A[3,,$IFP] = A[6,2] has position at byte 6, $IFP takes 6 as its value.

The following example shows how $IFP and $IFL (described later) are used:

```
A[1,1] = A[2,1]
A[$IFP, $IFL] = A[3,1]    "$IFP = 3, $IFL = 1"
A[3,$IFL] = A[2,,$IFP]    "$IFP = 3, $IFL = 2"
A[3,$IFL] = A[$IFP,,2]    "$IFP = 2, $IFL = 1"
```

$INPUT_FIELD_BIT_POS ($IFBP)

The $INPUT_FIELD_BIT_POS ($IFBP) function returns the last established position part of a source item. $IFBP includes only the bit position of the last source item referenced.

Examples In the assignment statement B[(5,$IFBP),4]=B[(5,2),4], $IFBP returns the value 2.

In the statement B[(5,$IFBP),4]=B[3,,(6,3)], $IFBP returns the value 1.

$INPUT_TRAILING_POS ($ITP)

The $INPUT_TRAILING_POS ($ITP) returns the position of the last byte in the current input record.

For example, suppose that the last field in the input record has a variable length. An assignment statement such as A[20,$IFL]=A[10,,$ITP] can be used to reformat the field into the output record. (The function $IFL gives the length of the source-item in bytes.)

$INPUT_TRAILING_BIT_POS ($ITBP)

The $INPUT_TRAILING_BIT_POS ($ITBP) function returns the position of the last bit in the current input record.

$MAX_OUTPUT_TRAILING_BIT_POS ($MOTBP)

The $MAX_OUTPUT_TRAILING_BIT_POS ($MOTBP) function returns the position of the last bit in the last byte of a maximum-length record. The maximum record length for the file is the value of its MAXIMUM_RECORD_LENGTH file attribute.

$MAX_OUTPUT_TRAILING_POS ($MOTP)

The $MAX_OUTPUT_TRAILING_POS ($MOTP) returns the maximum record length of the file. The maximum record length is the position of the last byte in the largest possible output record. It is determined by the MAXIMUM_RECORD_LENGTH attribute of the file.

For example, the destination item in the following assignment statement comprises the entire output record (from byte 1 through the maximum byte position).

```
A[1,,$MOTP] = A[1,,10]
```

$OUTPUT_TRAILING_POS ($OTP)

The $OUTPUT_TRAILING_POS ($OTP) returns the position of the last byte in the current output record (thus far built).

For example, suppose that the length of a destination item changes, as in A[2,$INPUT_FIELD_LENGTH] = A[25,,$INPUT_TRAILING_POSITION]. To use the position of the last byte in the output record thus far built, specify the statement A[$OTP + 1,$INPUT_FIELD_LENGTH] = A[$CURRENT_INPUT_POS,5]. Note that $OTP here is equivalent to $CURRENT_OUTPUT_POS-1.

$OUTPUT_TRAILING_BIT_POS ($OTBP)

The $OUTPUT_TRAILING_BIT_POS ($OTBP) function returns the position of the last bit in the last byte of the current output record.

Length Functions

Length functions are used in the length portion of a field descriptor to establish the length of a data field. Bit-count length functions are used to return length in bits in field descriptors having data type B.

$INPUT_FIELD_LENGTH ($IFL)

The $INPUT_FIELD_LENGTH ($IFL) function returns the length in bytes of the last source field referenced by a source-item.

Examples

```
A[1,$IFL] = A[9,8]              "IFL=8"
A[ , $IFL] = A[ , $IFL]         "IFL=8"

A[1,$IFL] = A[1,,$ITP]          "Here the output field
                                "assumes the length of the
                                "input record.

A[1, $IFL] = B[1,8]             "IFL=1"
```

$INPUT_FIELD_LENGTH_BITS ($IFLB)

The $INPUT_FIELD_LENGTH_BITS ($IFLB) returns the length part of the last source item in number of bits.

Examples In the statement B[4, $IFLB]=B[(5,2),,(6,1)], $IFLB returns the value 8.

$INPUT_RECORD_LENGTH ($IRL)

The $INPUT_RECORD_LENGTH ($IRL) function returns the current input record's length as a byte count.

Examples A[3,$IRL] = A[1,$IRL] takes a variable-length input record with alphanumeric data and reformats it as a variable-length output record.

$INPUT_RECORD_LENGTH_BITS ($IRLB)

The $INPUT_RECORD_LENGTH_BITS ($IRLB) function returns the current input record's length in number of bits.

Examples In the statement B[1, $IRLB]=B[1, , $ITBP], the value of $IRLB is the length of the input record in bits.

$OUTPUT_RECORD_LENGTH ($ORL)

The $OUTPUT_RECORD_LENGTH ($ORL) function returns the current output record's length as a byte count. The length is the byte count of the output record thus far built.

For example, the destination descriptor A[3,$ORL] uses the function as its length.

$OUTPUT_RECORD_LENGTH_BITS ($ORLB)

The $OUTPUT_RECORD_LENGTH_BITS ($ORLB) function returns the current output record's length in number of bits. The length value is the length of the record thus far built.

Examples B = [,$ORLB] doubles the size of the output record by appending data from the input record.

String Search Functions

You can use the input string position functions in the start-position and trailing-position parts of a field descriptor.

The input string position functions are as follows:

- $CURRENT_INPUT_STRING_POS
- $INPUT_STRING_POS

$CURRENT_INPUT_STRING_POS ($CISP)

The $CURRENT_INPUT_STRING_POS ($CISP) function returns the result of the last $INPUT_STRING_POS function. If none was executed, the value 1 is returned.

$INPUT_STRING_POS functions are evaluated in a field descriptor from left to right; where nested, they are evaluated inner to outer. Generally, field descriptors are evaluated from left to right. In assignment statements, however, the source item is evaluated before the destination item.

An example of how $CISP works is:

Example

```
A[3,$INPUT_FIELD_LENGTH] = ..
A[2,,$INPUT_STRING_POS('X',2 ,1 , FC)]
                                        "$ISP = 5, $IFL = 4"
A[10,2] = A[$CISP+1,2]                  "$CISP = 5, $CISP+1 = 6"
```

Input record: `0111XAB`

Output record: `111X AB`

$INPUT_STRING_POS searches for X, establishing the trailing position of a 4-byte field. 111X is placed into the 4-byte output field beginning at byte 3. The second statement places the string AB into the output record starting at byte 10.

$INPUT_STRING_POS ($ISP)

The $INPUT_STRING_POS ($ISP) searches for and establishes the position of a specified character string in an input record.

$INPUT_STRING_POS Format

This function differs in format from the other intrinsic functions. Its format is:

$INPUT_STRING_POS('string',*byte-position,occurrence,_ position-returned,_ default)*

You can replace the commas in the function format by one or more blanks.

string

ASCII string you are searching for. It cannot exceed 256 characters.

byte-position

Byte position at which the search is to begin. You specify position the same way as in the field descriptors, except that bit positions cannot be used. If you do not specify a value, $CURRENT_ INPUT_POS is used.

occurrence

Number of occurrences of the string to be found. If you do not specify a value, only the first occurrence is found.

position-returned

Indicates whether the function should return the byte position of the first or last character of the string.

FIRST_CHARACTER (FC)	First character (default)
LAST_CHARACTER (LC)	Last character

default

Arithmetic expression specifying the integer value the function is to return if it does not find the specified string occurrence.

In general, if this argument is omitted, the function returns an error when it cannot find the specified occurrence of the string. The exception is when the function is in a $IN_RECORD or $VALID_DATA function and the default argument is omitted. In that case, a failure to find the string occurrence does not return an error and processing continues.

How $INPUT_STRING_POS Works

$ISP searches the current input record from left to right for a specified string, beginning the search at a specified byte. The function searches for the desired occurrence of that string.

The function returns the position of the string as either the first character of the string or the last character, whichever you request on the function.

The $INPUT_STRING_POS function can be substituted in the field descriptor for an unknown starting or trailing position or both.

A failure by the $INPUT_STRING_POS function to find the string occurrence in a record returns an error except in these cases:

- A default value is specified on the $INPUT_STRING_POS function
- The $INPUT_STRING_POS function is used in a boolean function ($IN_RECORD or $VALID_DATA).

You can specify how you want file processing to continue after the error by the ERROR_DISPOSITION parameter on the SET_OUTPUT_ATTRIBUTES directive.

$INPUT_STRING_POS in Boolean Expressions

You can use a field descriptor containing $ISP in a relational expression of an IF statement to compare some aspect of input data. An example of a boolean relational expression containing $ISP is as follows:

```
IF A [$ISP('E', 1, 1, FC), 3] = 'EEE' THEN;
```

You can also use $ISP with an IF statement to test whether the field descriptor using the $ISP describes a field that exists in the record or whether data is valid for a particular data type. To apply that operation, use the $ISP in a field descriptor specified by a boolean function in the IF statement. For example:

```
IF $IN_RECORD(A [$ISP('D',1,1,FC), 1]) THEN;

IF $VALID_DATA(Z [$ISP ( ' = ' , 1 , 1 , FC ) +1,2] ) THEN;
```

$INPUT_STRING_POS Examples

Example A

```
A[1,3] = A[$ISP('C', 1, 2, LC)+1, 3]
```

Input record: ABABC1BABABC2BABABC3B

- $ISP searches for the second occurrence of character C, beginning at byte position 1. The +1 establishes start-position as the byte position after the last character in the string.
- $ISP is 12, because the second occurrence of C is at byte position 12; one more byte sets start-position at byte 13.
- Therefore, the data in the input field beginning at byte 13 with length of 3 bytes is placed in the output record starting at byte 1. The value in that field is 2BA.

Example B

```
a[$isp('2', 1,1,fc),1] = '9'
```

Input record: ABABC1BABABC2BABABC3B

- The $ISP replaces the string 2 with a 9. The input record is searched for the first occurrence of the string 2, beginning at byte 1. The position returned is the position of 2, which is byte 13.
- The literal 9 is entered in the output record at byte 13 of the output record.

Example C

```
A[1,1] = A[$INPUT_STRING_POS('C', ..
     $INPUT_STRING_POS('AB', 1, 2, ..
     LAST_CHARACTER) + 1, 1, LAST_CHARACTER) + 1 , , ..
     $INPUT_STRING_POS('B', $CURRENT_INPUT_STRING_POS, 1, ..
     FIRST_CHARACTER) -1]
```

Input record: ABABC1BABABC2BABABC3B

- The position of the source item is established by nested $INPUT_STRING_POS functions. The search for the first occurrence of C begins at the position established by the inner $INPUT_STRING_POS of the start-position part. The inner $INPUT_STRING_POS has searched for the second AB; the resulting byte index is the location of the last character in the string AB, and is byte 4.

- Thus, the outer $INPUT_STRING_POS begins its search from byte index 5, because 1 is added to the byte position established by the inner $INPUT_STRING_POS. The C is found immediately at byte position 5.

- Because of LAST_CHARACTER and +1, the start-position of the source item is established at byte 6. The $INPUT_STRING_POS in trailing-position establishes the length of the data item by trailing position. The search for the first occurrence of B starts from the $CURRENT_INPUT_STRING_POS value, which is 5. B is found at byte 7 but is excluded from the length calculation by the -1.

- The character that is written to the output record is 1.

Boolean Expressions

A boolean expression is an expression that is evaluated as true or false. Boolean expressions can contain:

- Relational expressions
- Boolean functions
- Boolean fields

A boolean expression can specify a logical value to be stored in a field or a condition to be tested by a CYCLE, EXIT, IF, REPEAT, or WHILE statement.

Relational Expressions

A relational expression has one of these formats:

source-item <relational operator> source-item

or

{ **intrinsic function** / **field descriptor** / **KEY** / **literal** / **integer constant** / **arithmetic expression** / **relational expression** } **<relational> operator** { **intrinsic function** / **field descriptor** / **KEY** / **literal** / **integer constant** / **arithmetic expression** / **relational expression** }

FMU ignores blanks; you can use them to improve readability.

Relational Operators

Relational operators are symbols that compare two values. The relational operators are as follows:

Operator	Meaning
<=	Less than or equal to
<	Less than
=	Equal to
<>	Not equal to
>	Greater than
>=	Greater than or equal to

Relational Expression Examples

Examples of relational expressions used in IF statements are:

```
IF N[12,3] < N[18,3] THEN;
```

Tests whether the value in the 3-byte normal integer field beginning at byte 12 is less than the value in the 3-byte field beginning at byte 18.

```
IF Z[1,3] = '+02' THEN;
```

Tests whether the input field has the value +02.

```
IF KEY <= '120' THEN;
```

Tests whether the value of the input key is less than or equal to 120.

Comparisons Between Data Types

Before FMU can compare the operands in a relational expression, it must reduce both operands to a common format. FMU performs the necessary conversion as indicated in this matrix:

	A	B	F	G	H	I	J	L	N	P	Q	U	Y	Z
A	A	Bs	F	F	H	H	J	L	H	H	H	H	H	H
B		Bs												
F		F	F											
G		F	F	F	F	F	F	L		F	F	F		
H		Bs	F		H	H	H	L		H/P	H/P	H/P		
I		Bs	F			I								
J		Bs	F		H	H	H	L		P	P	P		
L		L	L			L		L						
N		Bs	F	F	H	I	J	L	I	P	P	P		
P		Bs	F			P		L		P	P	P		
Q		Bs	F			P		L		P	P	P		
U		Bs	F			P		L		P	P	P		
Y		Bs	F	F	H	H	H	L	H	P	P	P	H	H
Z		Bs	F	F	H	H	H	L	H	P	P	P	H	H

Explanations of matrix notations:

Bs — Compared as an unsigned binary if both are positive; a negative field is always less than an unsigned binary field.

H/P — No conversion required.

F — Conversions during comparisons made using single or double precision floating point, depending on the largest number representable by either field.

Order of Precedence

The order of precedence of data types in a relational comparison is as follows:

1. L
2. F and G
3. H, J, P, Q, U, Y and Z
4. I and N
5. B
6. A

All data types except A and L are numeric. When an A field is compared to a numeric, the A field must be numeric.

If a numeric is compared to an L field, a zero value in the numeric field is taken as FALSE, and nonzero as TRUE. (This applies to NOS and NOS/BE files as well.)

Effect of Collation Tables

Regardless of whether a user-supplied collation table is provided for a NOS/VE file, all relationals evaluated in data type A comparison mode are ranked according to the ASCII collating sequence.

Boolean Functions

You can use a boolean function in place of a relational expression. A boolean function also yields a true or false result.

In a boolean function, a string search failure by $INPUT_STRING_POS results in a function evaluation of FALSE.

The boolean functions are:

- $IN_RECORD
- $VALID_DATA

$IN_RECORD ($IR)

An $IN_RECORD ($IR) function returns a TRUE value if the field is within the record. Otherwise, it returns a FALSE value.

Its format is:

$IN_RECORD(field descriptor)

The field descriptor specified in the function must follow the rules for valid field descriptors.

Examples of $IN_RECORD are shown in short IF statements:

```
IF $IN_RECORD(A[$ISP('D',1,1,FC),1]) THEN
 A[5,1] = A[$CISP,1]
IFEND
```

A simple search is done to find the 1-byte string D. If it is in the record, it is placed in the output record at byte position 5. Otherwise, the operation is skipped.

```
IF $IR(A[20,2]) THEN
 A[1,2] = A[20,2]
IFEND
```

This test determines whether there is a 2-byte alphanumeric field in the record at byte 20. If there is, it is placed in the output record; otherwise, the operation is skipped.

$VALID_DATA ($VD)

The $VALID_DATA ($VD) function returns a TRUE value if the data in the field is valid for the specified type; otherwise, it returns a FALSE value.

Its format is:

$VALID_DATA(field descriptor)

The field descriptor specified in the function must follow the rules for valid field descriptors.

An example of $VALID_DATA is shown as used in an IF statement:

```
IF $VALID_DATA(N[1,3]) THEN
     N[5,3] = N[1,3]
IFEND
```

This test determines whether the 3-byte field beginning at byte 1 contains a normal integer character. If it does, it is placed in the output record; otherwise, the operation is skipped.

In a boolean function, a string search failure by $INPUT_STRING_POS results in a function evaluation of FALSE.

Boolean Fields

A boolean field is a field descriptor that has a data type of L. It can be used in place of a boolean function.

The L data type is in binary format. That is, binary zero in the leftmost bit is defined as FALSE; a binary one in the leftmost bit is defined as TRUE.

Examples `IF L[5,4] THEN;  :IFEND`

The IF statement evaluates as TRUE or FALSE, depending on the contents of the field beginning at byte 5.

$INPUT_STRING_POS in Boolean Expressions

You can use an $INPUT_STRING_POS ($ISP) function in the field descriptors of boolean expressions. Failure of the function to find the string occurrence searched for depends on the type of boolean expression containing the function.

When $INPUT_STRING_POS is used in a relational expression, a string search failure is an error. The response to the error is determined by the ERROR_DISPOSITION parameter on the SET_OUTPUT_ATTRIBUTES directive.

When $INPUT_STRING_POS is used in a field descriptor of a boolean function, a string search failure is not an error.

Arithmetic Expressions

An FMU arithmetic expression is evaluated as an integer value. It can be used anywhere in a CREATE_OUTPUT_RECORD statement list to specify an integer value.

To evaluate an arithmetic expression, FMU performs arithmetic operations on integer operands.

The arithmetic operations, in order of precedence, are:

* /	Multiplication and division
mod	Modulus (remainder after division)
+ -	Addition and subtraction

The integer operands in an arithmetic expression can be specified as:

- Integer constants
- Literals
- Field descriptors
- The word KEY (to specify the primary-key field)
- Intrinsic functions
- Arithmetic expressions composed of the preceding elements and enclosed in parentheses

Operand values that are not integers are converted to integer. If this is not possible, an error is returned.

Floating point arithmetic is not supported. A value read by a field descriptor using the F or G data type is converted to integer before it is used.

Keyed File Reformatting 14

Keyed File Reformatting 14

A keyed file is a file whose organization allows record access by key value. The FILE_ORGANIZATION attribute of the file must specify one of the keyed-file organizations described in chapter 5 of this manual.

In most cases, you should use the COPY_KEYED_FILE command to copy keyed files. (The COPY_KEYED_FILE command is described in chapter 6 of this manual.) However, if record reformatting is required, you must use FMU to copy the keyed file.

Keyed Record Conversion

FMU can read records from a keyed file and/or write records to a keyed file.

The data type of a key is determined by the KEY_TYPE attribute value:

Key Type	Data Type
Collated (Symbolic)	A
Uncollated	A
Integer	I

Key Conversion Using FMU Command Copy

An FMU command copy results from an FMU command that specifies an input file and an output file, but no directives file. The key conversion performed by an FMU command copy depends on whether the keyed file is the input file, the output file, or both.

When the Input File is a Keyed File, But the Output File is Not:

Key conversion in this case depends on whether the input key is embedded or not, that is, it depends on the value of the EMBEDDED_KEY attribute.

EMBEDDED_KEY=TRUE for input file:

The data from the input record (including the key value) is copied to the output record.

EMBEDDED_KEY=FALSE for input file:

The data from the input record is copied to the output record, but the key value (because it is separate from the record) is discarded.

When the Output File is a Keyed File, But the Input File is Not:

In this case, the EMBEDDED_KEY attribute of the output file must be TRUE. The data from the input record is copied to the output record and the embedded primary key is defined by the key attributes of the output file.

If the EMBEDDED_KEY attribute of the output file is FALSE, an error results. This is because FMU cannot determine the value of the key because no key value is defined for the input record.

When Both the Input File and the Output File are Keyed Files:

FMU processing when both the input file and the output file are keyed files depends on the EMBEDDED_KEY attribute of the input file and of the output file.

Input Key Nonembedded, Output Key Nonembedded:

The nonembedded input key value becomes the nonembedded output key value. The input record data is copied to the output record.

Input Key Nonembedded, Output Key Embedded:

The nonembedded input key value is discarded. The input record data is copied to the output record. The output key is the value in the key field defined by the KEY_POSITION and KEY_LENGTH attributes of the output file.

Input Key Embedded, Output Key Nonembedded:

The embedded input key value becomes the nonembedded output key value. The input record data is copied to the output record. (The input key value is written twice: once as the nonembedded key and once in the record data.)

Input Key Embedded, Output Key Embedded:

The input record data is copied to the output record. The output key is the value in the key field defined by the KEY_POSITION and KEY_LENGTH attributes of the output file. (The KEY_POSITION and KEY_LENGTH attributes of the input file have no effect on the definition of the output key.)

Key Conversion Using CREATE_OUTPUT_RECORD Assignment Statements

To change the contents of the records in the new keyed file, you must use the CREATE_OUTPUT_RECORD directive. (For example, the fields of the record are reordered instead of merely being copied.)

You can use the CREATE_OUTPUT_RECORD assignment statements to transfer keys from input to output. The output key is initialized following the rules given for an FMU command copy and then the reformatting specified by the CREATE_OUTPUT_RECORD directive is performed.

You can use the keyword KEY to reference the input key or output key. In either case, the key can be embedded or nonembedded.

Examples of the use of the KEY operand are:

A [5,6] = KEY stores the key of the input record in the 6-byte field starting at position 5 of the output record.

KEY = A [1,5] stores the 5-byte input field in the area defined for the output key.

KEY = KEY places the value of the input key into the area defined for the output key.

Depending on the input file, use one of the following if you want to set the output key explicitly:

KEY = field descriptor

KEY = literal

KEY = KEY

If the key of the output file is embedded and the position, length, and FMU data type of the key are known, you can use a field descriptor to set the key; this practice, however, is not recommended.

FMU Examples 15

FMU Examples 15

This chapter contains simple examples of these FMU applications:

- Reformatting data
- Replacing occurrences of a string
- Creating an indexed-sequential file

Reformatting Data

The following example demonstrates some FMU functions: reordering data fields, using the $INPUT_STRING_POS to locate and reformat information, and adding a sequence number to each record.

If you are familiar with the FORM utility on NOS, this example is similar to the reformatting example in the FORM reference manual. Record selection, however, has not been included in the FMU example.

Input File Characteristics:

A file of student information, STDNTRF, is to be reformatted. The data in the input file STDNTRF is as follows:

Field	Position	Length
name	1-20	20
number	21-26	6
age	27-28	2
sex	29	1
dept	30-31	2
year	32	1
level	33-34	2
address	35-80	46

The records appear on the input file STDNTRF as follows:

```
Fender, A.T.     12345621M023JR123 A ST., Pozo, Calif.
Bender, R.L.     61234524F045SR456 North ST., Booneville, Calif.
Kettle, V.H.     56123422M034SR33 Main ST., San Jose, Calif.
McKee, N.R.      45612325F026SR2216 Bush ST., Bear Valley, Calif.
```

Output File Characteristics:

Name and number are to remain the same positionally. Age and sex are excluded from the new file. Year and department switch relative positions and are placed in the spots formerly occupied by age and sex.

Department 02 is to be changed to department 04. Level is excluded. Blanks are inserted.

Only the city part of the address will be extracted and placed on the new file.

The result is a 62-byte record.

The data fields for file GRADF are as follows:

Field	Position	Length
name	1-20	20
number	21-26	6
year	27	1
department	28-29	2
blank	30-32	3
city	33-62	30

The contents of the output file GRADF after the FMU run are as follows:

```
Fender, A.T.    1 123456304    Pozo
Bender, R.L.    2 612345504    Booneville
Kettle, V.H.    3 561234403    San Jose
McKee, N.R.     4 456123604    Bear Valley
```

The commands and the directive file for this job are as follows: (The / is a terminal prompt.)

```
/set_file_attribute file=stdntrf
/fmu, directives=stntdir, list=output
```

The directives in the directives file are:

```
set_input_attributes,file=stdntrf     "Specifies the input file
                                      "STDNTRF.

set_output_attributes,file=gradf      "Specifies the output file
                                      "GRADF.
```

```
set_sequence_attributes, ..          "Specifies sequence
file=gradf,sequence_field=N[18,2]    "numbering of records
                                     "beginning in byte position
                                     "18, which is part of the
                                     "name field of the output
                                     "record.

create_output_record,file=gradf ..   "Specifies the formatting of
                                     "the output record_preset_
                                     "value=character_blank
                                     "record.  The RECORD_
                                     "PRESET_VALUE parameter
                                     "indicates that the input
                                     "record fields that are
                                     "unreferenced by the CREOR
                                     "statements should be set
                                     "to blanks in the output
                                     "record.

  A[1,20] = A[1,20]                  "Specifies the formats of the
  N[21,6] = N[21,6]                  "first 27 bytes of the
  N[27,1] = N[32,1]                  "output record.

  if A[30,2] = '02' then             "Tests whether the input
       A[28,2] = '04'                "field at position 30 is a
  else                               "'02'.  If it is, '04' is
       A[28,2] = [30,2]              "stored in the department
  ifend                              "field; otherwise, the
                                     "contents of the input
                                     "field are stored in the
                                     "department field.

  A[30,3] = '   '                    "Stores three blanks in the
                                     "next three bytes.

                                     "This statement is
                                     "described after the
                                     "example.

A[33,29] = A[$input_string_pos(',', 35, 1, first_character) + ..
1, , $input_string_pos(',', $current_input_string_pos, 2, fc) ..
- 1]
```

```
creorend                    "Terminates the CREATE_
                            "OUTPUT_RECORD
                            "directive.
```

The final assignment statement extracts the city from the input address field and places it in the output record. The source item specifies that a $INPUT_STRING_POS search begins at position 35 to find the first occurrence of a comma (,). The comma is not to be carried into the output record so +1 is specified. The $INPUT_STRING_POS search for the second comma begins at the position returned by the last $ISP function. (The $CURRENT_INPUT_STRING_POS indicates the position at which the last comma was found.) The second comma is not to be carried into the output record so -1 is specified.

Replacing Occurrences of a String

This example demonstrates replacing a character or string with another character or string. The following elements are used:

- $IN_RECORD boolean function
- FOR statement for looping
- $CISP on the destination side of an assignment statement

The example input file SEMIF contains records that have semicolons scattered throughout. (The positions of the semicolons are not fixed.) FMU is to replace each semicolon with a dash (-).

FMU can replace a finite number of occurrences of character strings, but you must specify the upper limit on the number of occurrences. For this example, it was estimated that the semicolon occurs no more than 11 times in any one record.

The records in the input file SEMIF appear as follows:

```
AAAA;AAAAAAA
BBB;;BBB;BBB
;CCCCCCCC
DDDDDDDDDD
EEE;E;E;E;E;E
;;;;;;;;
FFFFFFFFF;
;GGGGGGGGG;
;;;;;HH;;;;;H
```

After the FMU run, the output file DASHF appears as follows:

```
AAAA-AAAAAAA
BBB--BBB-BBB
-CCCCCCCC
DDDDDDDDDD
EEE-E-E-E-E-E
----------;;;;;;
FFFFFFFFF-
-GGGGGGGGG-
-----HH-----H
```

As you can see, 11 iterations were not enough to replace all semicolons. Specifying 17 would have been sufficient.

The commands and the directive file for this job are as follows. (This example was run on a terminal; / is a prompt.)

```
/set_file_attribute file=dashf maximum_record_length=20
/fmu, directives=repldir, list=output
```

The directives in the directives file are as follows:

```
set_input_attributes,file=SEMIF

set_output_attributes,file=DASHF

create_output_record,file=DASHF, record_preset_value=character_
blank;

  "The RECORD_PRESET_VALUE parameter specifies that fields not
  "referenced in the CREOR assignment statements are to
  "duplicate the corresponding fields in the input record.

  if $in_record(A[ $input_string_pos( ';', 1, 1, fc),  1 ] ) then

      "The initial boolean function $IN_RECORD tests for the
      "existence of a semicolon.  The function's argument is a
      "field descriptor with a $INPUT_STRING_POS search function
      "in start-position.

      A[ $current_input_string_pos, 1 ] = '-'

          "If the $IN_RECORD encounters a semicolon, the semicolon
          "is replaced by a dash.  ($CISP indicates the position
          "returned by the last $ISP and designates the start-
          "position of the output field.)

          "After the first search is successful, a 10-iteration
          "FOR statement is entered.  Each successive search for
          "a semi-colon begins with the position returned by $CISP.
          "Also, the second occurrence relative to the last
          "occurrence is specified.

      for iterations = 1 to 10 do
          if $in_record( A[ $isp( ';', $cisp, 2, fc), 1 ] ) then
              A[ $cisp, 1 ] = '-'
          else
              stop
          ifend
      forend

          "All 10 iterations occur even if no semicolons follow
          "the first semicolon.  A $ISP search failure does not
          "return an error when used within the $IN_RECORD
          "function.

    ifend
create_output_record_end
```

Creating an Indexed-Sequential File

This example uses FMU to both reformat records and create an indexed sequential file with embedded keys.

The input file SEQFIL is a character data file; its records contain a 3-digit integer, 2 spaces, and a name as follows:

```
002  Church, Jared
001  Begelman, Nancy
020  Kowlski, Benjamin
003  Johnson, Tom
013  O'Toole, Donald
005  Williamson, Cary
007  Seger, Lawrence
011  Chang, Peter
014  Inouye, Leonard
006  Peterson, Lori
```

The example removes the spaces between the number and the name.

The interactive session is as follows: (/ , . ./ and ct? are prompts.)

```
"Defines the attributes of the
"new indexed sequential file.
/set_file_attributes file=isfil ..
../file_organization=indexed_sequential ..
../key_length=3 minimum_record_length=3 ..
../maximum_record_length=25
"Creates the directives file.
/collect_text output=directives
ct? set_input_attributes file=seqfil
ct? set_output_attributes file=isfil
ct? create_output_record file=isfil
ct?   a[1,3] = a[1,3]                        "Moves 3-digit number.
ct?   a[4,25] = a[6,$input_record_length]    "Moves name.
ct? create_output_record_end
ct? **
"Executes directives file.
/fmu directives=directives
```

To see that the file reformatting worked, you can enter a COPY_KEYED_FILE command to copy the file to $OUTPUT (if the file contains only displayable characters).

```
/copy_keyed_file input=$user.isfil output=$output
001Begelman, Nancy
002Church, Jared
003Johnson, Tom
005Williamson, Cary
006Peterson, Lori
007Seger, Lawrence
011Chang, Peter
013O'Toole, Donald
014Inouye, Leonard
020Kowlski, Benjamin
```

As you can see, the spaces between the number and the name are gone and the records are in sorted order by the primary key (the 3-digit number).

Appendixes

Glossary A

This appendix contains a glossary of terms listed in alphabetical order.

A

Advanced Access Methods (AAM)

The file management software that processes keyed files.

Alternate Index

An index built in a keyed file for an alternate key. The index associates each alternate-key value with a key list of one or more primary-key values.

Alternate Key

An optional key defined in addition to the primary key. An alternate key provides another method of directly accessing records in a keyed file. Unlike the primary key, an alternate key can be defined to allow duplicate values so that more than one record can have the same alternate-key value.

Alternate-Key Definition

The set of attributes that specify alternate-key characteristics. The alternate-key definition is used to build the alternate index for the key.

Ascending Sort Order

Ordering values from lowest to highest value. See Sort Order.

ASCII

American National Standard Code for Information Interchange. A 7-bit code representing a prescribed set of characters. NOS/VE stores each 7-bit ASCII code right-justified in an 8-bit byte.

B

Backup Copy

Copy kept for possible future recovery. Keyed-file backup copies should be written using the Backup_Permanent_File utility so they can be reloaded using the Recover_Keyed_File utility or the Restore_Permanent_File utility.

Basic Access Methods (BAM)

The file management software that processes sequential and byte-addressable files.

Beginning-of-Information (BOI)

The point at which file data begins. For a keyed file, the BOI file position means that the file is positioned to read the record with the lowest key value.

Bit

A binary digit. It has the value 0 or 1. See Byte.

Bit Index

The bit location relative to the first bit in a byte. NOS/VE bit positions start at bit 1 on the left and end at bit 8 on the right. NOS and NOS/BE bit positions start at bit 1 and end at bit 6.

Block

A logical or physical grouping of data. In a keyed file, blocks are units of file space linked by pointers.

Byte

A contiguous group of bits. A NOS/VE word has 8 bytes having 8 bits each. NOS/VE stores each ASCII character code in the rightmost 7 bits of a byte.

Byte-Addressable File Organization

A file organization in which records are accessed by their byte address in the file.

Byte Index

The byte position in a record relative to the beginning of the record.

C

Character

A letter, digit, space, or symbol represented by a code in a character set. The NOS/VE character set is the standard ASCII character set, so, unless stated otherwise, the term character in this manual refers to one of the 256 ASCII characters.

This page intentionally left blank.

Close Request

A program request notifying the system that the program no longer intends to access file data through the specified instance of open. In response, the system flushes all modified data from memory to the file and ends the connection between the program and the file.

Collated Key

The key type that orders key values according to a user-specified collation table. Contrast with Uncollated Key.

Collating Sequence

A set of values defining the collation weights of the 256 ASCII characters. The collation weights determine the sequence in which characters are ordered and their relative values when compared.

Collation Table

A data structure defining a collating sequence.

Collation Weight

The value assigned to a character that determines the position of that character when ordered using the collating sequence.

Command

A statement that initiates a specific operation. The SCL interpreter recognizes a command name if it is in the command list.

Command List

One or more entries that define the commands that are currently available.

Command Merge

A merge performed solely on the basis of MERGE command parameters.

Command Sort

A sort performed solely on the basis of SORT command parameters.

Command Utility

A NOS/VE program that adds its command list to the command list stack. It reads subcommands from the command file to determine its processing. Entry of a final subcommand (usually QUIT) ends command utility processing.

Concatenated Key

An alternate key that has two or more pieces. The pieces can be noncontiguous and can be concatenated in any order.

Cycle Reference

Specifies the specific cycle of a permanent file to be accessed. A cycle reference can be either an unsigned integer or one of the following designators: $HIGH, $LOW, $NEXT, $NEXT_LOW.

D

Data Block

A block in an indexed-sequential file in which data records are stored. Contrast with Index Block.

Data-Block Split

The process of creating two or three data blocks from an existing data block when a record to be written does not fit into the remaining space of the existing block.

Data Compression

The process of converting data so that it can be represented in less space. Usually, compressed data must be decompressed before it can be used.

Default Value

The value used for the parameter value if no value is explicitly specified.

Descending Sort Order

Ordering values from highest to lowest value. See Sort Order.

Destination Item

The description of an output field as used in an FMU CREATE_OUTPUT_RECORD directive assignment statement. It can be a field descriptor or KEY.

Direct-Access File Organization

A keyed-file organization in which each record is accessed directly by hashing its primary-key value. Records can be accessed sequentially, but the records are not returned in sorted order. Contrast with Indexed-Sequential File Organization.

Directive

A statement that specifies processing options for a command or subcommand. Both FMU and Sort/Merge interpret a set of directives. Their directives consist of a directive name followed by a parameter list.

Directive File

A file that contains only directives.

Display Code

A 6-bit character code used by NOS and NOS/BE systems.

Dual State Operations

Two operating systems executing simultaneously in the same mainframe. A CYBER 180 mainframe can execute NOS/VE and either NOS or NOS/BE.

Duplicate Key Value

The situation detected when a record to be written to the file has a key value that matches a key value already in the file (or another value for the alternate key in the same record). It can also be detected during application of a new alternate-key definition to a file.

Duplicate Key Value Control

The alternate-key attribute that indicates whether duplicate values are allowed for the key and, if so, how the duplicates are ordered.

E

EBCDIC

The abbreviation for extended binary-coded decimal interchange code, an 8-bit code representing a coded character set.

Embedded Key

Key that is part of the data in each record. (Alternate keys are always embedded.) Contrast with Nonembedded Key.

End-of-Information (EOI)

The point at which the data in a file ends. For a keyed file, the EOI file position means that the file is positioned after the record with the highest key value.

End-of-Partition (EOP)

A special delimiter in a file that uses the CDC variable (V) record type.

Exception Records File

A file to which invalid records are written before the records are removed from the process.

F

F Record Type

Fixed-length records, as defined by the ANSI standard.

Field

A subdivision of a record.

A field in a data record is defined by its position and length in the record. The location of a key value in a record is defined as a field.

A field in an SCL variable can be referenced by name. For example, the field NORMAL in a status record variable named OLD_STATUS is referenced as follows: OLD_STATUS.NORMAL

Field Descriptor

An FMU element that describes a data field in an input or output record in terms of data type, starting position, and field length.

File

A collection of information referenced by name. A file is an autonomous collection of information that exists separately from the programs that read or write the file.

SCL references an element consisting of a file path, an optional cycle reference (for permanent files), and a file position designator as follows:

```
file_path.cycle_reference.file_position
```

File Attribute

A characteristic of a file. The file attribute set defines the file structure and processing limitations.

File Cycle

A version of a file. All cycles of a file share the same file entry in a catalog. The file cycle is specified in a file reference by its number or by a special indicator, such as $NEXT.

File Organization

The file attribute that determines the record access method for the file. See Sequential File Organization, Byte-Addressable File Organization, and Keyed File Organization.

File Position

The current position of the file. The position at which the file is to be opened can be specified by the OPEN_POSITION file attribute or when specifying the file, using one of these designators:

$ASIS Leave the file in its current position.

$BOI Position the file at the beginning-of-information.

$EOI Position the file at the end-of-information.

This page intentionally left blank.

File Reference

An SCL element that identifies a file and optionally the file position to be established prior to the file's use. The format of a file reference is

:family.catalog.file.cycle.file_position

where catalog is one or more catalog names separated by a period.

where file is a 1- to 31-character name.

where cycle is a numeric value from 1 to 999 that represents a version of the file.

where file_position is one of the following:

$ASIS

$BOI

$EOI

See also File and File Position.

Floating-Point Number

A method of internal binary representation for numbers written with a decimal point; corresponds to a FORTRAN REAL or COBOL COMPUTATIONAL-1 number. Also stored as double precision, corresponding to FORTRAN DOUBLE PRECISION or COBOL COMPUTATIONAL-2.

Flush Request

A program request to write to the file device the parts of a file that have been modified in memory since the last time the file was written. For keyed files, the file device is always disk; for sequential files, the flush request can write to disk or to an interactive terminal.

G

Graphic

A character that can be printed or displayed.

H

Hashing Procedure

The procedure used to transform a primary-key value into a home block number in a direct-access file. The procedure is executed for each file request that specifies a key value.

Home Block

A unit of space in a direct-access file. If possible, data records are stored in home blocks. Contrast with Overflow Blocks.

I

Index Block

A block in an indexed-sequential file in which index records are stored. Contrast with Data Block.

Index-Block Split

The process of creating two index blocks from an existing index block when a record to be written does not fit into the remaining space of the existing block.

Index Level

A rank in the index-block hierarchy in an indexed-sequential file. To find the pointer to a data record, an index block must be searched for each index level.

Index Level Overflow

The condition when a record cannot be written to a file because writing the record would require addition of another index level and the file already has 15 index levels.

Index Record

A record in an index block that associates a key value with a pointer to either a data block or an index block in the next lower level of the index hierarchy.

Indexed-Sequential File Organization

A keyed-file organization in which records can be read sequentially ordered by key values or accessed individually by a key value.

Integer

Numeric data (positive or negative) that does not have any digits to the right of the assumed decimal point. An integer is stored internally as a binary value rather than a character value.

Integer Key

The key type that orders key values numerically. The key values can be positive or negative integers.

J

Job

A set of tasks executed for a user name. NOS/VE accepts interactive and batch jobs.

K

Key

For Sort/Merge, a key is a record field used to determine the position of the record within a sorted sequence of records.

In a keyed file, a key is a value associated with a record as a means of accessing the record. It may be a record field. See Primary Key and Alternate Key.

Key List

The sequence of primary-key values associated with an alternate-key value in an alternate index. If duplicate values are allowed for the key, a key list contains a primary-key value for each record in the file that contains the alternate-key value.

Key Type

The kind of data in a key.

For Sort/Merge, a key type is the name of a numeric data format or collating sequence.

For a keyed file, the possible key types are uncollated, collated, and integer.

Keyed-File Organization

A file organization that provides for record access by a key value. See Direct-Access File Organization and Indexed-Sequential File Organization.

Keyword

A word within a format that must be entered exactly as shown.

L

Literal

A symbol or quantity that is itself data rather than a reference to data. See Nonnumeric Literal and Numeric Literal.

Local File

A file that is accessed via the local catalog ($LOCAL). See also File, Path, and Local Path.

Local File Name

The name used by an executing job to reference a file while the file is assigned to the job's $LOCAL catalog. Only one file can be associated with a given name in one job; however, a file can have more than one instance of open in one job by that name.

Local Path

Identifies a local file as follows:

```
$LOCAL.file_name
```

Lock

A mechanism that makes a primary-key value (or, for a file lock, all primary-key values) inaccessible to other instances of open of the file.

Log

Entries recording a chronological series of events. The keyed-file interface uses the update recovery log. See also Update Recovery Log.

Login

The process used at a terminal to gain access to the system.

Logout

The process used to end a terminal session.

M

Major Sort Key

A sort key that is the most important key and is specified first. Sort/Merge uses this key before any other key. Contrast with Minor Sort Key.

This page intentionally left blank.

Mass Storage

A disk pack or rotating mass storage device; not a magnetic tape.

Media

Storage device on which data is recorded. Currently, NOS/VE files can be recorded on mass storage or magnetic tape.

Merge

The process of combining two or more presorted files.

Minor Sort Key

A sort key that is specified after the major sort key on a SORT or MERGE command or in a procedure call. Minor keys are sorted after the major sort key. Contrast with Major Sort Key.

Module

A unit of code. A CYBIL source code module is a compilation unit. An object module is the unit of object code corresponding to a compilation unit. A load module is a unit of object code stored in an object library.

N

Nested File

File defined within a keyed file. A nested file is recognized and used by the keyed-file interface; it is not recognized or used by the NOS/VE file system. When created, a keyed file contains one nested file, named $MAIN_FILE.

Nonembedded Key

A primary-key value that is not part of the record data. Contrast with Embedded Key.

Nonnumeric Literal

A literal bounded by quotation marks. A nonnumeric literal can include any character in the computer character set.

Null Suppression

Alternate-key attribute indicating that records with null alternate-key values are not included in the alternate index.

Numeric Literal

A literal composed of one or more numeric characters. A numeric literal can contain a decimal point, an algebraic sign, or both. A decimal point must not be the rightmost character; an algebraic sign must be the leftmost character.

O

Object Library

A library of modules that the system can load and execute as needed.

Operand

An entity to which an operation is applied.

Operator

The symbol that represents the action to be performed in an operation.

Overflow Block

Unit of space in a direct-access file used to store records whose home blocks are full. See also Home Block.

Owncode Procedure

A load module that Sort/Merge calls at a given point in its processing. An owncode procedure is called only if specified by an owncode procedure parameter on the SORT or MERGE command.

P

Packed Decimal

A numeric data format where each digit is represented by four bits, with two digits per standard 8-bit bytes.

Padding

Space deliberately left unused. Keyed-file blocks may be padded to allow easy insertion of records after creation of the file.

Parameter

A value list optionally preceded by and equivalenced to a parameter name. For example:

parameter name = value list

or

value list

Parameter List

A series of parameters separated by spaces or commas.

Parameter Name

A name that uniquely identifies a parameter.

Partition

A unit of data on a sequential or byte-addressable file delimited by end-of-partition separators or the beginning-of-information or end-of-information.

Path

Identifies a file. It may include the family name, user name, subcatalog name or names, file name, and cycle number.

Permanent File

A file preserved by NOS/VE across job executions and system deadstarts. A permanent file has an entry in a permanent catalog. See File.

Piece

One of the fields of a concatenated alternate key.

Position-Dependent Parameter

A parameter that must appear in a specified location, relative to other parameters. Contrast with Position-Independent Parameter.

Position-Independent Parameter

A parameter that consists of a parameter name followed by a value list. Contrast with Position-Dependent Parameter.

Primary Key

The required key in a keyed file. Primary-key values must be unique in the file. See also Alternate Key.

Procedure

A subroutine that passes values through its parameters; invoked when the name of the procedure is referenced in a CALL, ENTER, or subroutine calling statement.

Program Library List

The list of object libraries searched for modules during program loading. A program library list search is required to load a collation table module or an owncode procedure module.

R

Radix

Specifies the base of a number. NOS/VE recognizes binary, octal, decimal, and hexadecimal number bases. A radix enclosed in parentheses must follow a nondecimal number. The following numbers can be used to represent the radix:

2 Binary number base

8 Octal number base

10 Decimal number base

16 Hexadecimal number base

Random Access

The process of reading or writing a record directly without reading or writing the preceding records. Only disk files can be read or written randomly. Contrast with Sequential Access.

Real State

The CYBER 180 state executing the NOS or NOS/BE operating system. Contrast with Virtual State.

Record

A set of related data processed as a unit when reading or writing a file.

Recovery

Actions taken after damage occurs to alleviate the effects of the damage. Keyed-file recovery actions include reloading a backup copy and restoring the copy with an update recovery log.

Repeating Groups

An alternate-key attribute indicating that each data record can contain more than one value for the alternate key.

This page intentionally left blank.

Result Array

An array in which sort or merge statistics are returned.

Rewind

Operation that positions a file at its beginning-of-information.

Ring

The level of hardware protection given a file or segment. A file is protected from unauthorized access by tasks executing in higher rings.

Ring Attribute

A file attribute whose value consists of three ring numbers referenced as r1, r2, and r3. The ring numbers define four ring brackets for the file as follows:

Read bracket is 1 through r2.

Write bracket is 1 through r1.

Execute bracket is r1 through r2.

Call bracket is r2+1 through r3.

S

Sequential Access

The processing of records in order (physical or logical). Contrast with Random Access.

Sequential File Organization

A file organization in which records can only be processed in physical order. Records are always read in the order that they were written to the file.

Sign

Indicates whether a number is positive or negative. It can be denoted by the following characters:

+	Positive number
-	Negative number
space	Positive number

Signed Numeric Data

Integer data stored internally in ASCII code; sorted according to numeric order and sign of the integer the ASCII code represents.

Sort

The process of arranging records in a specified order.

Sort Key

A field of information within each record in a sort or merge input file that is used to determine the order in which records are written to the output file.

Sort Order

Ordering of data according to key fields, either ascending or descending.

Source Item

The description of the input field referenced in an FMU CREATE_OUTPUT_RECORD directive assignment statement or boolean relation. It can be a literal, field descriptor, or specific keyword.

Source Library

A collection of text units on a file generated and manipulated by the Source Code Utility (SCU).

Sparse-Key Control

An alternate-key attribute that allows only certain records to be included in the alternate index. Inclusion or exclusion of a record is determined by the character at the sparse-key control position of the record.

Statistics

Counts maintained for a keyed file. Each type of file access is counted as well as the number of records in the file.

Status Variable

An SCL variable of kind status that contains the completion status of a command.

Structural Properties

Characteristics of a keyed-file structure. The values of the characteristics change as the structure changes.

Sum Fields

A record field containing a numeric value which is added to the numeric value from the corresponding field of another record when the records are summed. The sum of the two values is stored in the new record that is created by the summing. See also Summing.

Summing

The process of combining two records having identical key values. The result of the process is a new record containing the original values of the key fields, the summed values of the sum fields, and data from one of the original records in any other record fields. See also Sum Fields.

System Command Language (SCL)

The block-structured interpretive language that provides the interface to the features and capabilities of NOS/VE. All commands and statements are interpreted by SCL before being processed by the system.

T

Task

The instance of execution of a program.

U

U Record Type

Records for which the record structure is undefined.

Uncollated Key

The key type that orders key values byte-by-byte according to the default ASCII collating sequence. Contrast with Collated Key.

Update Recovery Log

Log on which each backup or update operation to a keyed file is recorded so that, if the file is damaged, a backup file copy can be reloaded and updated using the information on the log.

V

V Record Type

Variable-sized records; system default record type. Each V-type record has a record header. The header contains the record length and the length of the preceding record.

Variable

Represents a data value.

SCL defines the following kinds of variables:

string boolean

integer status

Variable-Length Key

An alternate key whose values can vary in length up to the maximum key length defined for the key. A variable-length key can be defined as a single value in a record or as a sequence of values separated by one or more delimiter characters.

Virtual State

The CYBER 180 state executing the NOS/VE operating system. Contrast with Real State.

Related Manuals B

This appendix lists other Control Data manuals containing information related to the information given in this manual.

A complete list of NOS/VE manuals is given in the NOS/VE System Usage manual. If your site has installed the online manuals, you can find an abstract for each NOS/VE manual in the online System Information manual. To access the manual, enter the following NOS/VE command:

```
explain
```

Ordering Printed Manuals

To order a printed Control Data manual, send a completed order form to:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

To get an order form or more information about ordering Control Data manuals, write to the above address or call (612) 292-2101. If you are a Control Data employee, call (612) 292-2100.

Accessing Online Manuals

To access an online manual, log in to NOS/VE and supply the online title (listed in the following table) on the MANUAL parameter of an EXPLAIN command. For example, to see the NOS/VE Advanced File Management Usage manual, enter: explain, manual=afm

Table B-1. Related Manuals

Manual Title	Publication Number	Online Title
Advanced File Management:		
SCL for NOS/VE Advanced File Management Tutorial	60486412	AFM_T
SCL for NOS/VE Advanced File Management Summary	60486419	
NOS/VE Manuals:		
Introduction to NOS/VE Tutorial	60464012	
NOS/VE System Usage	60464014	
NOS/VE Source Code Management Usage	60464313	
NOS/VE Object Code Management Usage	60464413	
NOS/VE Commands and Functions	60464018	SCL
NOS/VE Diagnostic Messages	60464613	MESSAGES
NOS/VE User Validation	60464513	

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Title
Full Screen Editor for NOS/VE Tutorial/Usage	60464015	
CYBER Online Text for NOS/VE Usage	60488403	CONTEXT
Migration Manuals:		
Migration from NOS to NOS/VE Usage	60489503	MIGRATE_NOS
Migration from NOS to NOS/VE Standalone Usage	60489504	
Migration from NOS/BE to NOS/VE Usage	60489505	MIGRATE_ NOSBE
Migration from NOS/BE to NOS/VE Standalone Usage	60489506	
Migration from IBM to NOS/VE Usage	60489507	MIGRATE_IBM
Migration from VAX/VMS to NOS/VE Usage	60489508	MIGRATE_VAX
Other Related Manuals:		
COBOL for NOS/VE Usage	60486013	COBOL
CYBIL for NOS/VE Keyed-File and Sort/Merge Interfaces Usage	60464117	
FORTRAN for NOS/VE Language Definition Usage	60485913	
FORTRAN for NOS/VE Quick Reference		FORTRAN

ASCII Character Set C

Table C-1 lists the ASCII character set, the only character set used by NOS/VE.

NOS/VE supports the American National Standards Institute (ANSI) ASCII character set (ANSI X3.4-1977). Although the ASCII character set contains 256 character codes, only the first 128 codes are used; the second 128 codes are unassigned. NOS/VE represents each 7-bit ASCII code in an 8-bit byte. The 7 bits are right-justified in each byte. For ASCII characters, the eighth or leftmost bit is always zero.

Table C-1. ASCII Character Set

ASCII Code Decimal	ASCII Code Hexa-decimal	ASCII Code Octal	Graphic or Mnemonic	Name or Meaning
000	00	000	NUL	Null
001	01	001	SOH	Start of heading
002	02	002	STX	Start of text
003	03	003	ETX	End of text
004	04	004	EOT	End of transmission
005	05	005	ENQ	Enquiry
006	06	006	ACK	Acknowledge
007	07	007	BEL	Bell
008	08	010	BS	Backspace
009	09	011	HT	Horizontal tabulation
010	0A	012	LF	Line feed
011	0B	013	VT	Vertical tabulation
012	0C	014	FF	Form feed
013	0D	015	CR	Carriage return
014	0E	016	SO	Shift out
015	0F	017	SI	Shift in
016	10	020	DLE	Data link escape
017	11	021	DC1	Device control 1
018	12	022	DC2	Device control 2
019	13	023	DC3	Device control 3

(Continued)

Table C-1. ASCII Character Set *(Continued)*

ASCII Code Decimal	ASCII Code Hexa-decimal	ASCII Code Octal	Graphic or Mnemonic	Name or Meaning
020	14	024	DC4	Device control 4
021	15	025	NAK	Negative acknowledge
022	16	026	SYN	Synchronous idle
023	17	027	ETB	End of transmission block
024	18	030	CAN	Cancel
025	19	031	EM	End of medium
026	1A	032	SUB	Substitute
027	1B	033	ESC	Escape
028	1C	034	FS	File separator
029	1D	035	GS	Group separator
030	1E	036	RS	Record separator
031	1F	037	US	Unit separator
032	20	040	SP	Space
033	21	041	!	Exclamation point
034	22	042	"	Quotation marks
035	23	043	#	Number sign
036	24	044	$	Dollar sign
037	25	045	%	Percent sign
038	26	046	&	Ampersand
039	27	047	'	Apostrophe
040	28	050	(	Opening parenthesis
041	29	051	)	Closing parenthesis
042	2A	052	*	Asterisk
043	2B	053	+	Plus
044	2C	054	,	Comma
045	2D	055	-	Hyphen
046	2E	056	.	Period
047	2F	057	/	Slant

(Continued)

Table C-1. ASCII Character Set *(Continued)*

ASCII Code Decimal	ASCII Code Hexa-decimal	ASCII Code Octal	Graphic or Mnemonic	Name or Meaning
048	30	060	0	Zero
049	31	061	1	One
050	32	062	2	Two
051	33	063	3	Three
052	34	064	4	Four
053	35	065	5	Five
054	36	066	6	Six
055	37	067	7	Seven
056	38	070	8	Eight
057	39	071	9	Nine
058	3A	072	:	Colon
059	3B	073	;	Semicolon
060	3C	074	<	Less than
061	3D	075	=	Equals
062	3E	076	>	Greater than
063	3F	077	?	Question mark
064	40	100	@	Commercial at
065	41	101	A	Uppercase A
066	42	102	B	Uppercase B
067	43	103	C	Uppercase C
068	44	104	D	Uppercase D
069	45	105	E	Uppercase E
070	46	106	F	Uppercase F
071	47	107	G	Uppercase G
072	48	110	H	Uppercase H
073	49	111	I	Uppercase I
074	4A	112	J	Uppercase J
075	4B	113	K	Uppercase K

(Continued)

Table C-1. ASCII Character Set *(Continued)*

ASCII Code Decimal	ASCII Code Hexa-decimal	ASCII Code Octal	Graphic or Mnemonic	Name or Meaning
076	4C	114	L	Uppercase L
077	4D	115	M	Uppercase M
078	4E	116	N	Uppercase N
079	4F	117	O	Uppercase O
080	50	120	P	Uppercase P
081	51	121	Q	Uppercase Q
082	52	122	R	Uppercase R
083	53	123	S	Uppercase S
084	54	124	T	Uppercase T
085	55	125	U	Uppercase U
086	56	126	V	Uppercase V
087	57	127	W	Uppercase W
088	58	130	X	Uppercase X
089	59	131	Y	Uppercase Y
090	5A	132	Z	Uppercase Z
091	5B	133	[	Opening bracket
092	5C	134	\	Reverse slant
093	5D	135	]	Closing bracket
094	5E	136	^	Circumflex
095	5F	137	_	Underline
096	60	140	`	Grave accent
097	61	141	a	Lowercase a
098	62	142	b	Lowercase b
099	63	143	c	Lowercase c
100	64	144	d	Lowercase d
101	65	145	e	Lowercase e
102	66	146	f	Lowercase f
103	67	147	g	Lowercase g

(Continued)

Table C-1. ASCII Character Set *(Continued)*

ASCII Code Decimal	ASCII Code Hexa-decimal	ASCII Code Octal	Graphic or Mnemonic	Name or Meaning
104	68	150	h	Lowercase h
105	69	151	i	Lowercase i
106	6A	152	j	Lowercase j
107	6B	153	k	Lowercase k
108	6C	154	l	Lowercase l
109	6D	155	m	Lowercase m
110	6E	156	n	Lowercase n
111	6F	157	o	Lowercase o
112	70	160	p	Lowercase p
113	71	161	q	Lowercase q
114	72	162	r	Lowercase r
115	73	163	s	Lowercase s
116	74	164	t	Lowercase t
117	75	165	u	Lowercase u
118	76	166	v	Lowercase v
119	77	167	w	Lowercase w
120	78	170	x	Lowercase x
121	79	171	y	Lowercase y
122	7A	172	z	Lowercase z
123	7B	173	{	Opening brace
124	7C	174	|	Vertical line
125	7D	175	}	Closing brace
126	7E	176	~	Tilde
127	7F	177	DEL	Delete

Table C-1. ASCII Character Set *(Continued)*

ASCII Code				
Decimal	Hexa-decimal	Octal	Graphic or Mnemonic	Name or Meaning
112	70	160	p	Lowercase p
113	71	161	q	Lowercase q
114	72	162	r	Lowercase r
115	73	163	s	Lowercase s
116	74	164	t	Lowercase t
117	75	165	u	Lowercase u
118	76	166	v	Lowercase v
119	77	167	w	Lowercase w
120	78	170	x	Lowercase x
121	79	171	y	Lowercase y
122	7A	172	z	Lowercase z
123	7B	173	{	Opening brace
124	7C	174	|	Vertical line
125	7D	175	}	Closing brace
126	7E	176	~	Tilde
127	7F	177	DEL	Delete

Predecessor Product Comparison D

This appendix lists the major differences between the NOS/VE file management tools described in this manual and the products that performed similar functions on the NOS and NOS/BE operating systems.

NOS/VE Sort/Merge and Sort/Merge 5 Differences

This section lists the major differences between NOS/VE Sort/Merge and the Sort/Merge Version 5 that executes under NOS or NOS/BE.

NOS/VE Sort/Merge is compatible only with Sort/Merge Version 5; it does not attempt compatibility with any other Sort/Merge version.

NOS/VE Sort/Merge can only access NOS/VE disk files.

The File Management Utility (FMU) can convert NOS files into equivalent NOS/VE files. This utility converts the differences in byte size, collating sequence, record type, and block type.

Table D-1 compares Sort/Merge 5 with NOS/VE Sort/Merge.

Table D-1. Sort/Merge 5 to NOS/VE Sort/Merge Conversion

Subject	NOS Sort/Merge 5	NOS/VE Sort/Merge
Byte Size	6-bit byte	8-bit byte
Character Codes	Character data is internally represented in 6-bit display codes. character codes.	Character data is internally represented in 8-bit ASCII
Character Sets	Supports both the 63- and 64-character sets.	Supports only the 256-character ASCII set.
Collating Sequences	Five predefined collating sequences: ASCII6, COBOL6, DISPLAY, EBCDIC6, and INTBCD. ASCII6 is assumed if a sequence is not specified. A user-defined collating sequence has 64 positions. The collating sequences for Sort/Merge 5 and NOS/VE can have the same names but may have different collating sequences.	Six predefined collating sequences: ASCII, ASCII6, COBOL6, DISPLAY, EBCDIC, and EBCDIC6. ASCII is assumed if a sequence is not specified. Under NOS/VE a user-defined collating sequence has 256 positions. (NOS/VE Sort/Merge can use the SEQR parameter to fill the rest).
Direct Processing	Sort/Merge reads and writes directly (instead of through CYBER Record Manager) if so specified by the FASTIO parameter or the SM5FAST procedure.	NOS/VE Sort/Merge does not support this option (all records are read and written through the access method).

(Continued)

Table D-1. Sort/Merge 5 to NOS/VE Sort/Merge Conversion *(Continued)*

Subject	NOS Sort/Merge 5	NOS/VE Sort/Merge
Error File	The default error file is the listing file.	The default error file is $ERRORS.
Error Messages	Sort/Merge 5 error numbers and message text do not follow NOS/VE error message conventions.	NOS/VE Sort/Merge error numbers and message text follow NOS/VE error message conventions.
	The Sort/Merge 5 error messages are listed in the Sort/Merge 5 Reference Manual.	The NOS/VE Sort/Merge error messages are listed in the NOS/VE Diagnostic Messages manual.
Estimated Number of Records	Value used as specified on the ENR parameter.	Value can be specified on the ENR parameter, but the value is not used.
Exception File Processing	Does not perform exception file processing.	Performs exception file processing if an exception file is specified for the sort or merge.
File Attributes	The NOS default file attributes are valid for a sort or merge.	The NOS/VE default value for the minimum record length attribute could cause a fatal error if no key field was specified for the sort or merge.

(Continued)

Table D-1. Sort/Merge 5 to NOS/VE Sort/Merge Conversion *(Continued)*

Subject	NOS Sort/Merge 5	NOS/VE Sort/Merge
File Manipulation	Files are rewound before and after use, depending on the type of file or unless a FILE control statement parameter specifies otherwise.	Files are not rewound by Sort/ Merge. The open position of a NOS/VE file is determined by the value of its open_position attribute.
Interactive Prompting	Parameters can be entered in response to an interactive dialogue.	Interactive prompting is not currently implemented.
Listing File	Does not provide a parameter to specify the name of the listing file.	Provides the LIST parameter to specify the listing file.
	The default listing file is file OUTPUT.	The default listing file is file $LIST.
Messages	Under Sort/Merge 5, messages were written to the dayfile.	Under NOS/VE Sort/Merge, messages are written to the list and error files.
Owncode Procedures	The file containing the compiled owncode routines is specified by the OWNF parameter.	Any owncode procedures specified for a sort or merge must be accessible from an object library in the current object library list.
		Owncode procedure names must be specified using uppercase letters only unless C170_ COMPATIBLE=TRUE is specified.

(Continued)

Table D-1. Sort/Merge 5 to NOS/VE Sort/Merge Conversion *(Continued)*

Subject	NOS Sort/Merge 5	NOS/VE Sort/Merge
Parameter Names	Parameter names do not follow the SCL parameter name convention.	The full form of the NOS/VE Sort/Merge parameter names follow the SCL parameter name convention. An abbreviated form is also available that matches the NOS Sort/Merge 5 parameter names.
Parameter Positional Order	The parameter positions are as listed in the Sort/Merge 5 Reference Manual; the parameter order differs from NOS/VE.	The parameter positions are as listed in chapter 3 of this manual; the parameter order differs from NOS.
Parameters for NOS/VE Only	Not available for Sort/Merge 5.	The parameters LIST_OPTIONS, LOAD_COLLATING_TABLE, and RESULT_ARRAY. (NOS/VE only)
Retain Parameter	The parameter values YES and NO can be abbreviated as Y and N, respectively.	The parameter values YES and NO cannot be abbreviated.
Signed Overpunches	Twenty overpunches are defined.	Thirty-four overpunches are defined.
Sort Command Format	The sort command begins with the word SORT5 followed by a period.	The sort command begins with the word SORT followed by a space or a comma.

(Continued)

Table D-1. Sort/Merge 5 to NOS/VE Sort/Merge Conversion *(Continued)*

Subject	NOS Sort/Merge 5	NOS/VE Sort/Merge
Status Parameter	The STATUS parameter specifies a variable that is set to a value representing the highest level of error.	The NOS/VE status parameter specifies a status variable in which the completion status of the command or procedure is returned.
Zero Comparison	Negative zero is ordered before positive zero.	Positive and negative zero are ordered equally.

Keyed-File Utilities Comparison

This section compares the utilities provided by the CYBER Record Manager Advanced Access Methods (CRM AAM) with the NOS/VE keyed-file utilities.

FLBLOK Utility

The FLBLOK utility suggests an appropriate block size for a keyed file. Using the estimates provided by the FLBLOK control statement parameters, the utility can make a better block size selection than the CRM open routines that do not have that information.

NOS/VE does not require a separate utility to suggest a block size because the estimates (such as the average record length) are specified as file attribute values and are therefore available to the open procedures.

MIPGEN Utility

The MIPGEN utility defines and deletes alternate keys. It uses a directive file as input.

The CREATE_ALTERNATE_INDEXES command utility defines and deletes alternate keys for NOS/VE keyed files. It uses subcommands as input.

MIPDIS Utility

The MIPDIS utility disassociates index and data files.

NOS/VE stores indexes and data in the same file so there is no need for a utility to disassociate associated files.

Key Analysis Utility

The key analysis is used to analyze a hashing routine for use with a direct-access file.

NOS/VE does not currently support a key analysis utility for its direct-access file organization.

CREATE Utility

The CREATE utility can be called from a program to create a direct access file.

You can create a NOS/VE direct-access file from within a program or by using the SCL commands SET_FILE_ATTRIBUTES and COPY_KEYED_FILE.

FORM and FMU Comparison

This section shows similarities and differences between FMU on NOS/VE and FORM on NOS.

Table D-2 shows the directive counterparts between FMU and FORM.

Table D-3 compares the syntax and capabilities of the directives CREATE_OUTPUT_RECORD (of FMU) and REF (of FORM).

Table D-4 gives comparisons of other directive elements, primarily relating to the CREATE_OUTPUT_RECORD (CREOR) and REF directives.

Table D-5 compares the handling of keys for keyed files.

Table D-2. Counterparts of the FMU and FORM Directives

Function	FMU	FORM
Specifying the input file	SET_INPUT_ATTRIBUTES (SETFA)	INP
Specifying the output file	SET_OUTPUT_ATTRIBUTES (SETOA)	OUT
Converting an IBM file	See Migration from IBM to NOS/VE manual	CON
Selecting records	Not supported in this release	QAL
Reformatting a file	CREATE_OUTPUT_RECORD (CREOR)	REF
Sequence numbering a file	SET_SEQUENCE_ ATTRIBUTES (SETSA)	SEQ
Format printing	SET_PRINT_ATTRIBUTES	PAG
Specifying subroutines	Ability to use subroutines not supported in this release	XEQ

Table D-3. Comparison of CREOR and REF Directives

Function	FMU CREOR	FORM REF	Comments
Data reformatting	Assignment statement	Reformat item	Similar
Iterative capability	FOR statement that allows statements to be processed in a loop	n(simple-re-format) n(reformat--string)	Similar capability
Conditional processing	IF statement and relational expression	Selector expression	Similar; FORM has advantage with expressions, including compound relational expressions.
Nested conditional	Nested IFs are allowed	Seven nesting levels with selector expressions	FMU has added strength with no upper limit on nesting.
Branching	IF with ELSEIF allows n+2 branches	Limited to two	FMU can test on same data field for many cases and branch out.
Stopping record reformatting	STOP statement	Q specificatio-n	Similar

Table D-4. Functional FMU and FORM Comparisons

Function	FMU	FORM	Comments
Data descriptors	Field descriptors	Item descriptors	Similar; FMU descriptors allow for field length in terms of trailing position.
If byte index is omitted in the data descriptors when reformatting	Pointer moves to next field; applies to source and destination items	Pointer moves to next field; applies to source and destination items	
Search features; can be used to locate string when reformatting data of variable length hand position	$INPUT_STRING_POS function	Search descriptors	Similar; $ISP is in function format.
Nested searching	$ISP can be nested	None	FMU has added strength.
Referencing pointer after search	$CISP function	None	
Pointer referencing functions	Intrinsic Functions; See chapter 4	None	FMU has added strength.
Reference forward or backward from current position	Intrinsic Function + n	+ n for current position	FMU has added strength.

Function	FMU	FORM	Comments
Data descriptors	Field descriptors	Item descriptors	Similar; FMU descriptors allow for field length in terms of trailing position.
Duplicating directive specifications	DUPLICATE_SPECIFICATION (or DS) parameter on most directives	Equating logical file names	
Conditional testing:	Boolean functions		
For existence of field	$IN_RECORD	None	
For validity of field content	$VALID_DATA	None	
Disposition of fields not referenced during reformatting	RECORD_PRESET_VALUE parameter of the CREOR directive	BGD parameter of the OUT directive	Similar.
File rewinding	No parameter; handled by other means	REW parameter of the INP and OUT directives	

Table D-5. Handling Keys for Indexed Sequential Files

Function	FMU	FORM	Comments
Insert input key into output record	KEY used as source item in assignment statement	KEY source item used in reformat item	Similar.
Save actual key for different output file	Actual key files not supported in this release	KEYA source item used with reformat item	
Designate key in output record	KEY used as destination item in assignment statement	KEY parameter in OUT directive	KEY (FMU) receives the input key or field, whereas KEY (FORM) designates only the postion and type of field in the output record for nonembedded keys.

Collation Tables E

This appendix describes how to use a collation table to specify how a key is ordered.

The collation table can be one of the NOS/VE predefined collation tables (listed at the end of this appendix) or a user-defined collation table.

The key to be ordered can be one of the following:

- A sort key. You can associate a key type name with the collation table using the Sort/Merge parameter LOAD_COLLATING_TABLE described in chapter 2. The key type can then be used in a key field definition.
- The primary key of a keyed file. You specify the collation-table name as the value of the COLLATE_TABLE_NAME file attribute when creating the file (as described in chapter 6).
- An alternate key of a keyed file. You specify the collation-table name as the value of the COLLATE_TABLE_NAME attribute of the alternate-key definition (as described in chapter 7).

Using NOS/VE Predefined Collation Tables

To use one of the NOS/VE predefined collation tables listed at the end of this appendix, you specify the name of the predefined collation table as the collation-table name. Unlike user-defined collation table modules, use of NOS/VE predefined collation tables does not require the addition of an object library to the program-library list.

Sort/Merge Example:

To use the predefined collation table OSV$EBCDIC to define the key type MY_KEY_TYPE, you would specify this Sort/Merge parameter:

```
load_collating_table=(my_key_type,osv$ebcdic)
```

Then, to define the first 10 bytes of the record as a key field to be sorted in ascending order using the key type, you would specify this Sort/Merge parameter:

```
key=((1,10,,my_key_type,a))
```

Keyed-File Example:

To use the predefined collation table OSV$EBCDIC to order the primary key of a new keyed file, you specify the key type as collated and the collate-table name as OSV$EBCDIC as follows:

```
/set_file_attribute file=new_keyed_file ..
../file_organization=indexed_sequential ..
../maximum_record_length=100 ..
../key_length=10 minimum_record length=10 ..
../key_type=collated collate_table_name=osv$ebcdic
```

FMU Example:

To use the predefined collation table OSV$EBCDIC to evaluate relational expressions when FMU formats records for an output file, specify the CONDITION_COLLATING_SEQUENCE parameter on the SET_OUTPUT_ATTRIBUTES directive for the output file.

For example:

```
set_output_attributes file=output_file_1 ..
  condition_collating_sequence=ebcdic
```

Notice that the directive specifies EBCDIC, instead of OSV$EBCDIC. The FMU keywords for the predefined NOS/VE collation tables do not use the OSV$ prefix.

Using User-Defined Collation Tables

You can use any collation table stored in an object-library file if you have permission to read the file. To use the collation table, you perform these steps:

1. Add the object library to your program-library list using a SET_PROGRAM_ATTRIBUTES command, such as:

```
set_program_attribute add_library=$user.object_library
```

2. Specify the collation-table name. (The name must be in the entry-point list of the object library as displayed by a DISPLAY_OBJECT_LIBRARY command.)

The process of storing a collation table in an object library is described in the Creating a Collation Table section.

For the purposes of these examples, assume another user has given you permission to read an object library file named .WIZARD.OBJECT_LIBRARY containing a collation table. The entry point for the collation table is named CASE_INSENSITIVE.

Sort/Merge Example:

To use the CASE_INSENSITIVE collation table:

1. Add the object library to your program-library list before entering the SORT or MERGE command:

```
/set_program_attribute add_library=.wizard.object_library
```

   On the SORT or MERGE command, specify the collation table as a key type and use the key type in a key-field definition:

```
/sort from=unsorted_file to=sorted_file ..
../load_collating_table=(my_key_type,case_insensitive) ..
../key=((1..24,my_key_type,d))
```

2. Keyed-File Example:

   To use the CASE_INSENSITIVE collation table to order a new alternate key of a keyed file:

   a. Add the object library to your program-library list before entering the CREATE_ALTERNATE_INDEXES command:

      ```
      /set_program_attribute add_library=.wizard.object_library
      ```

   b. Begin a CREATE_ALTERNATE_INDEXES utility session. On the CREATE_KEY_DEFINITION subcommand, specify the collated key type and the collation-table name:

      ```
      creai/create_key_definition key_name=alternate_key_1 ..
      creai../key_position=0 key_length=24 key_type=collated ..
      creai../collate_table_name=case_insensitive
      ```

Creating a Collation Table

Besides using collation tables created by others, you can also create your own collation tables. The process of using your collation tables was described previously under Using User-Defined Collation Tables.

Creating your own collation table involves these steps:

1. Writing a source code module to initialize the collation table.
2. Compiling the source code module to create the object module.
3. Storing the object module in an object library.

Writing a Module to Initialize a Collation Table

A module to initialize a collation table must perform these steps:

1. Declare a 256-integer array.
2. Store an integer (0 through 255) in each element of the array.

The values stored in the array are the collating weights. The collating weight in an array element is the collating weight assigned to the ASCII character corresponding to that element.

How a Collation Table Works

To determine the correct values with which to initialize the collation table, you must understand how a collation table works.

As shown in figure E-1, each element in the collation table corresponds to an 8-bit character code. The first 128 elements correspond to the 128 characters in the ASCII character set (as listed in appendix B). For example, the element 0 in the table corresponds to the NUL character (character code 00 decimal). Element 65 corresponds to the A character (character code 65 decimal).

Figure E-2 shows how a collation table is initialized for the default ASCII collating sequence. As you can see, the element rank matches the element contents. For example, the element for character NUL (character code 00) contains 0. The element for character A (character code 65) contains 65.

Now, suppose we change two values in the initialized collation table in figure E-2. We change the A element to contain 66 (B) and the B element to contain 65 (A). This collating sequence would order all B characters as A characters and all A characters as B characters. A sort using the collating sequence would sort all B characters before all A characters.

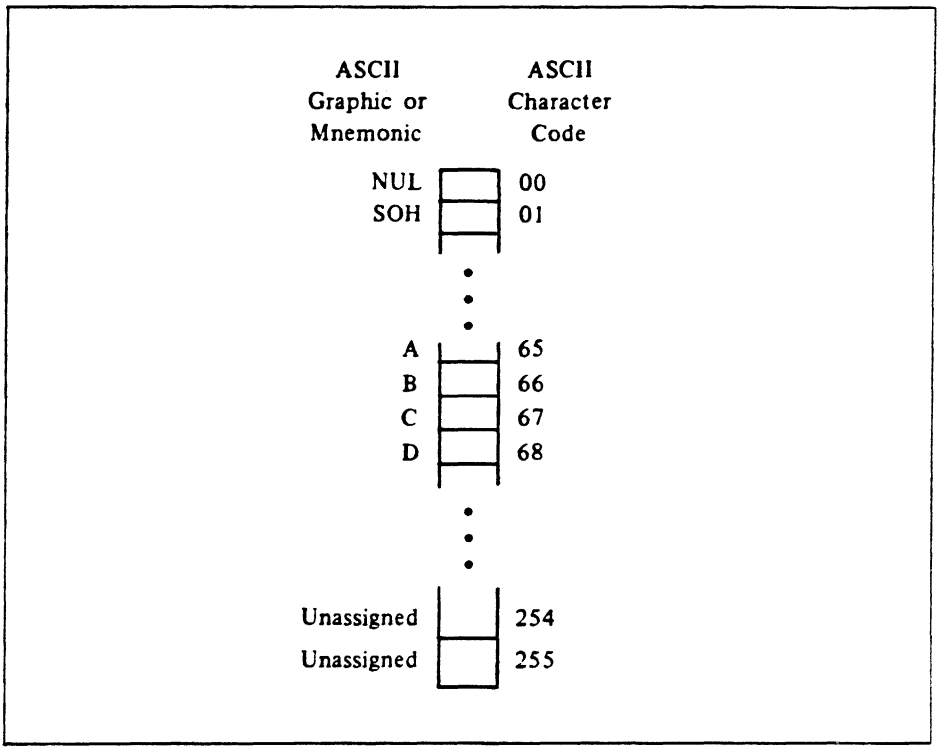


Figure E-1. Uninitialized Collation Table

Or, suppose we change the initialized collation table so that the A element contains 65 (A) and the B element also contains 65 (A). This collating sequence would order all A characters as A characters and all B characters as A characters. A sort using the collating sequence would intermix A and B characters.

NOTE

Be careful when choosing the collating sequence to order the primary key of a keyed file. A collating sequence that assigns equal values to different characters reduces the possible unique key values.

If the key values are collated equally, the values are no longer unique in the file. For example, if B is collated as A, the key value B is a duplicate of key value A.

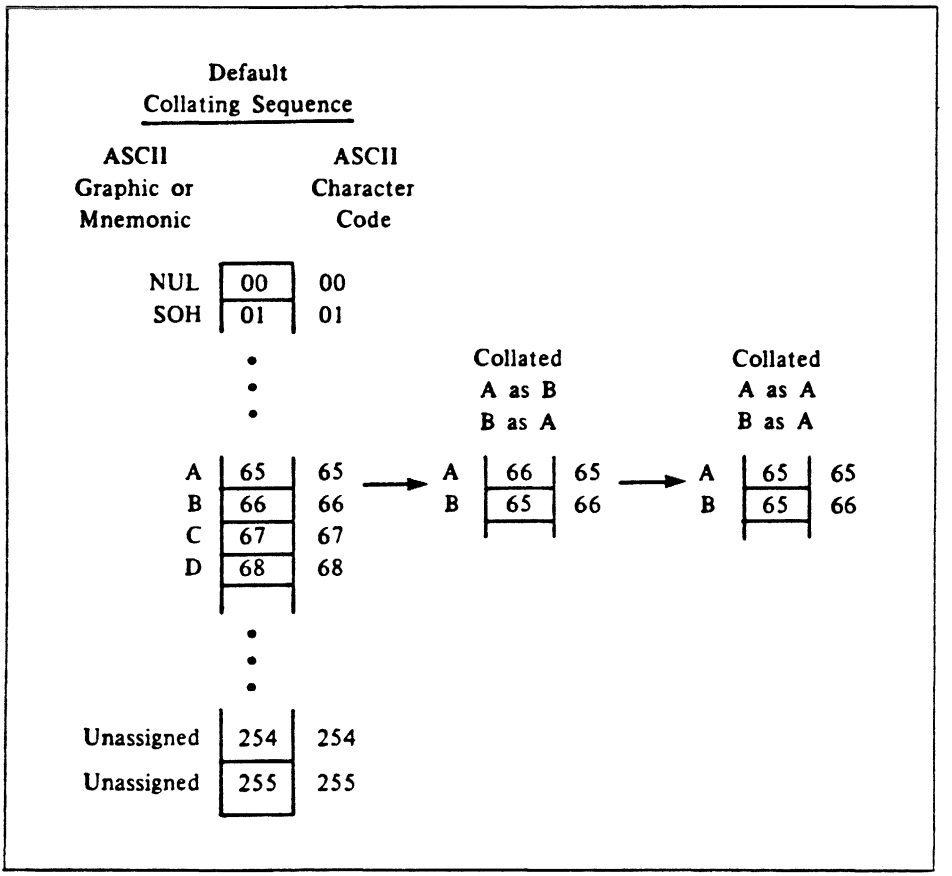


Figure E-2. Collation Table Initialized to the Default ASCII Collating Sequence

CYBIL Collation Table Initialization Examples

A CYBIL module to initialize a collation table declares a 256-element array variable and assigns a value to each element.

NOTE

The array variable must be assigned the XDCL attribute so that the name is an entry point to the module. A module can define more than one collation table by declaring and initializing more than one XDCL array variable.

Figure E-3 shows a CYBIL module named MY_MODULE that initializes an XDCL variable named CASE_INSENSITIVE. It assigns the collating weight for the space character (32) to all elements except the elements corresponding to letters. Each lowercase letter is to be ordered the same as the corresponding uppercase letter (a the same as A, b the same as B, and so forth).

```
MODULE my_module;

VAR
case_insensitive: [STATIC,READ,XDCL] ARRAY [CHAR] OF 0..255 :=

[ { Collating weights for the first 65 non-letter characters }
  REP 65 OF 32,

{ Collating weights for the uppercase letters }
  {A} 65, {B} 66, {C} 67, {D} 68, {E} 69, {F} 70, {G} 71,
  {H} 72, {I} 73, {J} 74, {K} 75, {L} 76, {M} 77, {N} 78,
  {O} 79, {P} 80, {Q} 81, {R} 82, {S} 83, {T} 84, {U} 85,
  {V} 86, {W} 87, {X} 88, {Y} 89, {Z} 90,

{ Collating weights for the next 6 non-letter characters }
REP 6 OF 32,

{ Collating weights for the lowercase letters }
  {a} 65, {b} 66, {c} 67, {d} 68, {e} 69, {f} 70, {g} 71,
  {h} 72, {i} 73, {j} 74, {k} 75, {l} 76, {m} 77, {n} 78,
  {o} 79, {p} 80, {q} 81, {r} 82, {s} 83, {t} 84, {u} 85,
  {v} 86, {w} 87, {x} 88, {y} 89, {z} 90,

{ Collating weights for the last 133 non-letter characters }
  REP 133 OF 32 ];

 MODEND;
```

Figure E-3. CASE_INSENSITIVE Collating Sequence Initialization Module

Sort/Merge Example:

If Sort/Merge used the collation table from figure E-3, it would sort characters as follows:

Unordered: 10]garbageGARBAGEgarbage9815];]

Ordered: 10]9815];]aaAAaabBbeEeggGGggrRr

Keyed-File Example:

If a keyed file used the collation table from figure E-3, all nonalphabetic key values would be duplicates. Uppercase and lowercase letters would be collated the same, so the key value ABCD would be a duplicate of the key value abcd.

Storing a Module in an Object Library

Source module compilation writes an object module on an object file. You then use the NOS/VE command utility CREATE_OBJECT_LIBRARY to create an object library containing the module. (The CREATE_OBJECT_LIBRARY utility is described in detail in the NOS/VE Object Code Management Usage manual.)

For this example, assume that you have written a CYBIL module (such as the one in figure E-3) to initialize a collation table and that your source text is on file $USER.SOURCE. The following commands compile the program and then store the module on file $USER.COLLATION_LIBRARY.

```
/cybil input=$user.source binary_object=object_file ..
.. /list=list_file
/create_object_library
COL/add_module library=object_file
COL/generate_library library=$user.collation_library
COL/quit
```

NOS/VE Predefined Collation Table Listings

The collating sequences of the predefined collation tables are listed in tables E-1 through E-11.

Several of the predefined collation tables have two variants, FOLDED and STRICT. The variants FOLDED and STRICT indicate two different mappings of the characters not in the 63 or 64 characters of the original CYBER 170 collating sequence.

- A strict mapping maps all characters not in the original 64- or 63-character set to the space character.
- A folded mapping maps some of these characters to the space character, but not others. (For the exact mapping, see the collating sequence in the table.)

The predefined collation tables are for these collating sequences:

Collating Sequence	Predefined Collation Table
CYBER 170 FTN5 default	OSV$ASCII6_FOLDED and OSV$ASCII6_STRICT
CYBER 170 COBOL5 default	OSV$COBOL6_FOLDED and OSV$COBOL6_STRICT
CYBER 170 63-character display code	OSV$DISPLAY63_FOLDED and OSV$DISPLAY63_STRICT
CYBER 170 64-character display code	OSV$DISPLAY64_FOLDED and OSV$DISPLAY64_STRICT
Full EBCDIC	OSV$EBCDIC
	EBCDIC 6-bit subset supported by CYBER 170 COBOL5 and SORT5 OSV$EBCDIC_FOLDED and OSV$EBCDIC_STRICT

Sort/Merge uses predefined collation tables for its predefined collating sequences as follows:

Key Type	Predefined Collation Table
ASCII6	OSV$ASCII6_FOLDED
COBOL6	OSV$COBOL6_FOLDED
DISPLAY	OSV$DISPLAY64_FOLDED
EBCDIC	OSV$EBCDIC
EBCDIC6	OSV$EBCDIC6_FOLDED

The Sort/Merge key type ASCII uses the default ASCII collating sequence; that is, it orders the ASCII character set as it is listed in appendix C.

Table E-1. OSV$ASCII6_FOLDED Collating Sequence

The ASCII codes not listed in this table (ASCII codes 0 through 1F and 7F through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	20	SP	Space
01	21	!	Exclamation point
02	22	"	Quotation marks
03	23	#	Number sign
04	24	$	Dollar sign
05	25	%	Percent sign
06	26	&	Ampersand
07	27	'	Apostrophe
08	28	(	Opening parenthesis
09	29	)	Closing parenthesis
10	2A	*	Asterisk
11	2B	+	Plus
12	2C	,	Comma
13	2D	-	Hyphen
14	2E	.	Period
15	2F	/	Slant
16	30	0	Zero
17	31	1	One
18	32	2	Two
19	33	3	Three
20	34	4	Four
21	35	5	Five
22	36	6	Six
23	37	7	Seven

(Continued)

Table E-1. OSV$ASCII6_FOLDED Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	38	8	Eight
25	39	9	Nine
26	3A	:	Colon
27	3B	;	Semicolon
28	3C	<	Less than
29	3D	=	Equals
30	3E	>	Greater than
31	3F	?	Question mark
32	40,60	@,`	Commercial at, grave accent
33	41,61	A,a	Uppercase A, lowercase a
34	42,62	B,b	Uppercase B, lowercase b
35	43,63	C,c	Uppercase C, lowercase c
36	44,64	D,d	Uppercase D, lowercase d
37	45,65	E,e	Uppercase E, lowercase e
38	46,66	F,f	Uppercase F, lowercase f
39	47,67	G,g	Uppercase G, lowercase g
40	48,68	H,h	Uppercase H, lowercase h
41	49,69	I,i	Uppercase I, lowercase i
42	4A,6A	J,j	Uppercase J, lowercase j
43	4B,6B	K,k	Uppercase K, lowercase k
44	4C,6C	L,l	Uppercase L, lowercase l
45	4D,6D	M,m	Uppercase M, lowercase m
46	4E,6E	N,n	Uppercase N, lowercase n
47	4F,6F	O,o	Uppercase O, lowercase o
48	50,70	P,p	Uppercase P, lowercase p
49	51,71	Q,q	Uppercase Q, lowercase q
50	52,72	R,r	Uppercase R, lowercase r
51	53,73	S,s	Uppercase S, lowercase s

(Continued)

Table E-1. OSV$ASCII6_FOLDED Collating Sequence
(Continued)

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	54,74	T,t	Uppercase T, lowercase t
53	55,75	U,u	Uppercase U, lowercase u
54	56,76	V,v	Uppercase V, lowercase v
55	57,77	W,w	Uppercase W, lowercase w
56	58,78	X,x	Uppercase X, lowercase x
57	59,79	Y,y	Uppercase Y, lowercase y
58	5A,7A	Z,z	Uppercase Z, lowercase z
59	5B,7B	[,{	Opening bracket, opening brace
60	5C,7C	\,|	Reverse slant, vertical line
61	5D,7D	],}	Closing bracket, closing brace
62	5E,7E	^,~	Circumflex, tilde
63	5F	_	Underline

Table E-2. OSV$ASCII6_STRICT Collating Sequence

The ASCII codes not listed in this table (ASCII codes 0 through 1F and 60 through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	20	SP	Space
01	21	!	Exclamation point
02	22	"	Quotation marks
03	23	#	Number sign
04	24	$	Dollar sign
05	25	%	Percent sign
06	26	&	Ampersand
07	27	'	Apostrophe
08	28	(	Opening parenthesis
09	29	)	Closing parenthesis
10	2A	*	Asterisk
11	2B	+	Plus
12	2C	,	Comma
13	2D	-	Hyphen
14	2E	.	Period
15	2F	/	Slant
16	30	0	Zero
17	31	1	One
18	32	2	Two
19	33	3	Three
20	34	4	Four
21	35	5	Five
22	36	6	Six
23	37	7	Seven

(Continued)

Table E-2. OSV$ASCII6_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	38	8	Eight
25	39	9	Nine
26	3A	:	Colon
27	3B	;	Semicolon
28	3C	<	Less than
29	3D	=	Equals
30	3E	>	Greater than
31	3F	?	Question mark
32	40	@	Commercial at
33	41	A	Uppercase A
34	42	B	Uppercase B
35	43	C	Uppercase C
36	44	D	Uppercase D
37	45	E	Uppercase E
38	46	F	Uppercase F
39	47	G	Uppercase G
40	48	H	Uppercase H
41	49	I	Uppercase I
42	4A	J	Uppercase J
43	4B	K	Uppercase K
44	4C	L	Uppercase L
45	4D	M	Uppercase M
46	4E	N	Uppercase N
47	4F	O	Uppercase O
48	50	P	Uppercase P
49	51	Q	Uppercase Q
50	52	R	Uppercase R
51	53	S	Uppercase S

(Continued)

Table E-2. OSV$ASCII6_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	54	T	Uppercase T
53	55	U	Uppercase U
54	56	V	Uppercase V
55	57	W	Uppercase W
56	58	X	Uppercase X
57	59	Y	Uppercase Y
58	5A	Z	Uppercase Z
59	5B	[	Opening bracket
60	5C	\	Reverse slant
61	5D	]	Closing bracket
62	5E	^	Circumflex
63	5F	_	Underline

Table E-3. OSV$COBOL6_FOLDED Collating Sequence

Any ASCII codes not listed in this table (ASCII codes 0 through 1F and 7F through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	20	SP	Space
01	40,60	@,`	Commercial at, grave accent
02	25	%	Percent sign
03	5B,7B	[,{	Opening bracket, opening brace
04	5F	_	Underline
05	23	#	Number sign
06	26	&	Ampersand
07	27	'	Apostrophe
08	3F	?	Question mark
09	3E	>	Greater than
10	5C,7C	\,|	Reverse slant, vertical line
11	5E,7E	^,~	Circumflex, tilde
12	2E	.	Period
13	29	)	Closing parenthesis
14	3B	;	Semicolon
15	2B	+	Plus
16	24	$	Dollar sign
17	2A	*	Asterisk
18	2D	-	Hyphen
19	2F	/	Slant
20	2C	,	Comma
21	28	(	Opening parenthesis
22	3D	=	Equals
23	22	"	Quotation marks

(Continued)

Table E-3. OSV$COBOL6_FOLDED Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	3C	<	Less than
25	41,61	A,a	Uppercase A, lowercase a
26	42,62	B,b	Uppercase B, lowercase b
27	43,63	C,c	Uppercase C, lowercase c
28	44,64	D,d	Uppercase D, lowercase d
29	45,65	E,e	Uppercase E, lowercase e
30	46,66	F,f	Uppercase F, lowercase f
31	47,67	G,g	Uppercase G, lowercase g
32	48,68	H,h	Uppercase H, lowercase h
33	49,69	I,i	Uppercase I, lowercase i
34	21	!	Exclamation point
35	4A,6A	J,j	Uppercase J, lowercase j
36	4B,6B	K,k	Uppercase K, lowercase k
37	4C,6C	L,l	Uppercase L, lowercase l
38	4D,6D	M,m	Uppercase M, lowercase m
39	4E,6E	N,n	Uppercase N, lowercase n
40	4F,6F	O,o	Uppercase O, lowercase o
41	50,70	P,p	Uppercase P, lowercase p
42	51,71	Q,q	Uppercase Q, lowercase q
43	52,72	R,r	Uppercase R, lowercase r
44	5D,7D	],}	Closing bracket, closing brace
45	53,73	S,s	Uppercase S, lowercase s
46	54,74	T,t	Uppercase T, lowercase t
47	55,75	U,u	Uppercase U, lowercase u
48	56,76	V,v	Uppercase V, lowercase v
49	57,77	W,w	Uppercase W, lowercase w
50	58,78	X,x	Uppercase X, lowercase x
51	59,79	Y,y	Uppercase Y, lowercase y

(Continued)

Table E-3. OSV$COBOL6_FOLDED Collating Sequence
(Continued)

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	5A,7A	Z,z	Uppercase Z, lowercase z
53	3A	:	Colon
54	30	0	Zero
55	31	1	One
56	32	2	Two
57	33	3	Three
58	34	4	Four
59	35	5	Five
60	36	6	Six
61	37	7	Seven
62	38	8	Eight
63	39	9	Nine

Table E-4. OSV$COBOL6_STRICT Collating Sequence

The ASCII codes not listed in this table (ASCII codes 0 through 1F and 60 through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	20	SP	Space
01	40	@	Commercial at
02	25	%	Percent sign
03	5B	[	Opening bracket
04	5F	_	Underline
05	23	#	Number sign
06	26	&	Ampersand
07	27	'	Apostrophe
08	3F	?	Question mark
09	3E	>	Greater than
10	5C	\	Reverse slant
11	5E	^	Circumflex
12	2E	.	Period
13	29	)	Closing parenthesis
14	3B	;	Semicolon
15	2B	+	Plus
16	24	$	Dollar sign
17	2A	*	Asterisk
18	2D	-	Hyphen
19	2F	/	Slant
20	2C	,	Comma
21	28	(	Opening parenthesis
22	3D	=	Equals
23	22	"	Quotation marks

(Continued)

Table E-4. OSV$COBOL6_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	3C	<	Less than
25	41	A	Uppercase A
26	42	B	Uppercase B
27	43	C	Uppercase C
28	44	D	Uppercase D
29	45	E	Uppercase E
30	46	F	Uppercase F
31	47	G	Uppercase G
32	48	H	Uppercase H
33	49	I	Uppercase I
34	21	!	Exclamation point
35	4A	J	Uppercase J
36	4B	K	Uppercase K
37	4C	L	Uppercase L
38	4D	M	Uppercase M
39	4E	N	Uppercase N
40	4F	O	Uppercase O
41	50	P	Uppercase P
42	51	Q	Uppercase Q
43	52	R	Uppercase R
44	5D	]	Closing bracket
45	53	S	Uppercase S
46	54	T	Uppercase T
47	55	U	Uppercase U
48	56	V	Uppercase V
49	57	W	Uppercase W
50	58	X	Uppercase X
51	59	Y	Uppercase Y

(Continued)

Table E-4. OSV$COBOL6_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	5A	Z	Uppercase Z
53	3A	:	Colon
54	30	0	Zero
55	31	1	One
56	32	2	Two
57	33	3	Three
58	34	4	Four
59	35	5	Five
60	36	6	Six
61	37	7	Seven
62	38	8	Eight
63	39	9	Nine

Table E-5. OSV$DISPLAY63_FOLDED Collating Sequence

The ASCII codes not listed in this table (ASCII codes 0 through 1F, 25, and 7F through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	41,61	A,a	Uppercase A, lowercase a
01	42,62	B,b	Uppercase B, lowercase b
02	43,63	C,c	Uppercase C, lowercase c
03	44,64	D,d	Uppercase D, lowercase d
04	45,65	E,e	Uppercase E, lowercase e
05	46,66	F,f	Uppercase F, lowercase f
06	47,67	G,g	Uppercase G, lowercase g
07	48,68	H,h	Uppercase H, lowercase h
08	49,69	I,i	Uppercase I, lowercase i
09	4A,6A	J,j	Uppercase J, lowercase j
10	4B,6B	K,k	Uppercase K, lowercase k
11	4C,6C	L,l	Uppercase L, lowercase l
12	4D,6D	M,m	Uppercase M, lowercase m
13	4E,6E	N,n	Uppercase N, lowercase n
14	4F,6F	O,o	Uppercase O, lowercase o
15	50,70	P,p	Uppercase P, lowercase p
16	51,71	Q,q	Uppercase Q, lowercase q
17	52,72	R,r	Uppercase R, lowercase r
18	53,73	S,s	Uppercase S, lowercase s
19	54,74	T,t	Uppercase T, lowercase t
20	55,75	U,u	Uppercase U, lowercase u
21	56,76	V,v	Uppercase V, lowercase v
22	57,77	W,w	Uppercase W, lowercase w
23	58,78	X,x	Uppercase X, lowercase x

(Continued)

Table E-5. OSV$DISPLAY63_FOLDED Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	59,79	Y,y	Uppercase Y, lowercase y
25	5A,7A	Z,z	Uppercase Z, lowercase z
26	30	0	Zero
27	31	1	One
28	32	2	Two
29	33	3	Three
30	34	4	Four
31	35	5	Five
32	36	6	Six
33	37	7	Seven
34	38	8	Eight
35	39	9	Nine
36	2B	+	Plus
37	2D	-	Hyphen
38	2A	*	Asterisk
39	2F	/	Slant
40	28	(	Opening parenthesis
41	29	)	Closing parenthesis
42	24	$	Dollar sign
43	3D	=	Equals
44	20	SP	Space
45	2C	,	Comma
46	2E	.	Period
47	23	#	Number sign
48	5B,7B	[,{	Opening bracket, opening brace
49	5D,7D	],}	Closing bracket, closing brace
50	3A	:	Colon
51	22	"	Quotation marks

(Continued)

Table E-5. OSV$DISPLAY63_FOLDED Collating Sequence
(Continued)

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	5F	_	Underline
53	21	!	Exclamation point
54	26	&	Ampersand
55	27	'	Apostrophe
56	3F	?	Question mark
57	3C	<	Less than
58	3E	>	Greater than
59	40,60	@,`	Commercial at, grave accent
60	5C,7C	\,|	Reverse slant, vertical line
61	5E,7E	^,~	Circumflex, tilde
62	3B	;	Semicolon

Table E-6. OSV$DISPLAY63_STRICT Collating Sequence

The ASCII codes not listed in this table (ASCII codes 0 through 1F, 25, and 60 through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	41	A	Uppercase A
01	42	B	Uppercase B
02	43	C	Uppercase C
03	44	D	Uppercase D
04	45	E	Uppercase E
05	46	F	Uppercase F
06	47	G	Uppercase G
07	48	H	Uppercase H
08	49	I	Uppercase I
09	4A	J	Uppercase J
10	4B	K	Uppercase K
11	4C	L	Uppercase L
12	4D	M	Uppercase M
13	4E	N	Uppercase N
14	4F	O	Uppercase O
15	50	P	Uppercase P
16	51	Q	Uppercase Q
17	52	R	Uppercase R
18	53	S	Uppercase S
19	54	T	Uppercase T
20	55	U	Uppercase U
21	56	V	Uppercase V
22	57	W	Uppercase W
23	58	X	Uppercase X

(Continued)

Table E-6. OSV$DISPLAY63_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	59	Y	Uppercase Y
25	5A	Z	Uppercase Z
26	30	0	Zero
27	31	1	One
28	32	2	Two
29	33	3	Three
30	34	4	Four
31	35	5	Five
32	36	6	Six
33	37	7	Seven
34	38	8	Eight
35	39	9	Nine
36	2B	+	Plus
37	2D	-	Hyphen
38	2A	*	Asterisk
39	2F	/	Slant
40	28	(	Opening parenthesis
41	29	)	Closing parenthesis
42	24	$	Dollar sign
43	3D	=	Equals
44	20	SP	Space
45	2C	,	Comma
46	2E	.	Period
47	23	#	Number sign
48	5B	[	Opening bracket
49	5D	]	Closing bracket
50	3A	:	Colon
51	22	"	Quotation marks

(Continued)

Table E-6. OSV$DISPLAY63_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	5F	_	Underline
53	21	!	Exclamation point
54	26	&	Ampersand
55	27	'	Apostrophe
56	3F	?	Question mark
57	3C	<	Less than
58	3E	>	Greater than
59	40	@	Commercial at
60	5C	\	Reverse slant
61	5E	^	Circumflex
62	3B	;	Semicolon

Table E-7. OSV$DISPLAY64_FOLDED Collating Sequence

The ASCII codes not listed in this table (ASCII codes 0 through 1F and 60 through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	3A	:	Colon
01	41,61	A,a	Uppercase A, lowercase a
02	42,62	B,b	Uppercase B, lowercase b
03	43,63	C,c	Uppercase C, lowercase c
04	44,64	D,d	Uppercase D, lowercase d
05	45,65	E,e	Uppercase E, lowercase e
06	46,66	F,f	Uppercase F, lowercase f
07	47,67	G,g	Uppercase G, lowercase g
08	48,68	H,h	Uppercase H, lowercase h
09	49,69	I,i	Uppercase I, lowercase i
10	4A,6A	J,j	Uppercase J, lowercase j
11	4B,6B	K,k	Uppercase K, lowercase k
12	4C,6C	L,l	Uppercase L, lowercase l
13	4D,6D	M,m	Uppercase M, lowercase m
14	4E,6E	N,n	Uppercase N, lowercase n
15	4F,6F	O,o	Uppercase O, lowercase o
16	50,70	P,p	Uppercase P, lowercase p
17	51,71	Q,q	Uppercase Q, lowercase q
18	52,72	R,r	Uppercase R, lowercase r
19	53,73	S,s	Uppercase S, lowercase s
20	54,74	T,t	Uppercase T, lowercase t
21	55,75	U,u	Uppercase U, lowercase u
22	56,76	V,v	Uppercase V, lowercase v
23	57,77	W,w	Uppercase W, lowercase w

(Continued)

Table E-7. OSV$DISPLAY64_FOLDED Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	58,78	X,x	Uppercase X, lowercase x
25	59,79	Y,y	Uppercase Y, lowercase y
26	5A,7A	Z,z	Uppercase Z, lowercase z
27	30	0	Zero
28	31	1	One
29	32	2	Two
30	33	3	Three
31	34	4	Four
32	35	5	Five
33	36	6	Six
34	37	7	Seven
35	38	8	Eight
36	39	9	Nine
37	2B	+	Plus
38	2D	-	Hyphen
39	2A	*	Asterisk
40	2F	/	Slant
41	28	(	Opening parenthesis
42	29	)	Closing parenthesis
43	24	$	Dollar sign
44	3D	=	Equals
45	20	SP	Space
46	2C	,	Comma
47	2E	.	Period
48	23	#	Number sign
49	5B,7B	[,{	Opening bracket, opening brace
50	5D,7D	],}	Closing bracket, closing brace
51	25	%	Percent sign

(Continued)

Table E-7. OSV$DISPLAY64_FOLDED Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	22	"	Quotation marks
53	5F	_	Underline
54	21	!	Exclamation point
55	26	&	Ampersand
56	27	'	Apostrophe
57	3F	?	Question mark
58	3C	<	Less than
59	3E	>	Greater than
60	40,60	@,`	Commercial at, grave accent
61	5C,7C	\,|	Reverse slant, vertical line
62	5E,7E	^,~	Circumflex, tilde
63	3B	;	Semicolon

Table E-8. OSV$DISPLAY64_STRICT Collating Sequence

The ASCII codes not listed in this table (ASCII codes 0 through 1F and 60 through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	3A	:	Colon
01	41	A	Uppercase A
02	42	B	Uppercase B
03	43	C	Uppercase C
04	44	D	Uppercase D
05	45	E	Uppercase E
06	46	F	Uppercase F
07	47	G	Uppercase G
08	48	H	Uppercase H
09	49	I	Uppercase I
10	4A	J	Uppercase J
11	4B	K	Uppercase K
12	4C	L	Uppercase L
13	4D	M	Uppercase M
14	4E	N	Uppercase N
15	4F	O	Uppercase O
16	50	P	Uppercase P
17	51	Q	Uppercase Q
18	52	R	Uppercase R
19	53	S	Uppercase S
20	54	T	Uppercase T
21	55	U	Uppercase U
22	56	V	Uppercase V
23	57	W	Uppercase W

(Continued)

Table E-8. OSV$DISPLAY64_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	58	X	Uppercase X
25	59	Y	Uppercase Y
26	5A	Z	Uppercase Z
27	30	0	Zero
28	31	1	One
29	32	2	Two
30	33	3	Three
31	34	4	Four
32	35	5	Five
33	36	6	Six
34	37	7	Seven
35	38	8	Eight
36	39	9	Nine
37	2B	+	Plus
38	2D	-	Hyphen
39	2A	*	Asterisk
40	2F	/	Slant
41	28	(	Opening parenthesis
42	29	)	Closing parenthesis
43	24	$	Dollar sign
44	3D	=	Equals
45	20	SP	Space
46	2C	,	Comma
47	2E	.	Period
48	23	#	Number sign
49	5B	[	Opening bracket
50	5D	]	Closing bracket
51	25	%	Percent sign

(Continued)

Table E-8. OSV$DISPLAY64_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	22	"	Quotation marks
53	5F	_	Underline
54	21	!	Exclamation point
55	26	&	Ampersand
56	27	'	Apostrophe
57	3F	?	Question mark
58	3C	<	Less than
59	3E	>	Greater than
60	40	@	Commercial at
61	5C	\	Reverse slant
62	5E	^	Circumflex
63	3B	;	Semicolon

Table E-9. OSV$EBCDIC Collating Sequence

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
000	00	NUL	Null
001	01	SOH	Start of heading
002	02	STX	Start of text
003	03	ETX	End of text
004	9C	—	Unassigned
005	09	HT	Horizontal tabulation
006	86	—	Unassigned
007	7F	DEL	Delete
008	97	—	Unassigned
009	8D	—	Unassigned
010	8E	—	Unassigned
011	0B	VT	Vertical tabulation
012	0C	FF	Form feed
013	0D	CR	Carriage return
014	0E	SO	Shift out
015	0F	SI	Shift in
016	10	DLE	Data link escape
017	11	DC1	Device control 1
018	12	DC2	Device control 2
019	13	DC3	Device control 3
020	9D	—	Unassigned
021	85	—	Unassigned
022	08	BS	Backspace
023	87	—	Unassigned
024	18	CAN	Cancel
025	19	EM	End of medium
026	92	—	Unassigned
027	8F	—	Unassigned
028	1C	FS	File separator
029	1D	GS	Group separator
030	1E	RS	Record separator
031	1F	US	Unit separator

(Continued)

Table E-9. OSV$EBCDIC Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
032	80	—	Unassigned
033	81	—	Unassigned
034	82	—	Unassigned
035	83	—	Unassigned
036	84	—	Unassigned
037	0A	LF	Line feed
038	17	ETB	End of transmission block
039	1B	ESC	Escape
040	88	—	Unassigned
041	89	—	Unassigned
042	8A	—	Unassigned
043	8B	—	Unassigned
044	8C	—	Unassigned
045	05	ENQ	Enquiry
046	06	ACK	Acknowledge
047	07	BEL	Bell
048	90	—	Unassigned
049	91	—	Unassigned
050	16	SYN	Synchronous idle
051	93	—	Unassigned
052	94	—	Unassigned
053	95	—	Unassigned
054	96	—	Unassigned
055	04	EOT	End of transmission
056	98	—	Unassigned
057	99	—	Unassigned
058	9A	—	Unassigned
059	9B	—	Unassigned
060	14	DC4	Device control 4
061	15	NAK	Negative acknowledge
062	9E	—	Unassigned
063	1A	SUB	Substitute

(Continued)

Table E-9. OSV$EBCDIC Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
064	20	SP	Space
065	A0	—	Unassigned
066	A1	—	Unassigned
067	A2	—	Unassigned
068	A3	—	Unassigned
069	A4	—	Unassigned
070	A5	—	Unassigned
071	A6	—	Unassigned
072	A7	—	Unassigned
073	A8	—	Unassigned
074	5B	[	Opening bracket
075	2E	.	Period
076	3C	<	Less than
077	28	(	Opening parenthesis
078	2B	+	Plus
079	21	!	Exclamation point
080	26	&	Ampersand
081	A9	—	Unassigned
082	AA	—	Unassigned
083	AB	—	Unassigned
084	AC	—	Unassigned
085	AD	—	Unassigned
086	AE	—	Unassigned
087	AF	—	Unassigned
088	B0	—	Unassigned
089	B1	—	Unassigned
090	5D	]	Closing bracket
091	24	$	Dollar sign
092	2A	*	Asterisk
093	29	)	Closing parenthesis
094	3B	;	Semicolon
095	5E	^	Circumflex

(Continued)

Table E-9. OSV$EBCDIC Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
096	2D	-	Hyphen
097	2F	/	Slant
098	B2	—	Unassigned
099	B3	—	Unassigned
100	B4	—	Unassigned
101	B5	—	Unassigned
102	B6	—	Unassigned
103	B7	—	Unassigned
104	B8	—	Unassigned
105	B9	—	Unassigned
106	7C	|	Vertical line
107	2C	,	Comma
108	25	%	Percent sign
109	5F	_	Underline
110	3E	>	Greater than
111	3F	?	Question mark
112	BA	—	Unassigned
113	BB	—	Unassigned
114	BC	—	Unassigned
115	BD	—	Unassigned
116	BE	—	Unassigned
117	BF	—	Unassigned
118	C0	—	Unassigned
119	C1	—	Unassigned
120	C2	—	Unassigned
121	60	`	Grave accent
122	3A	:	Colon
123	23	#	Number sign
124	40	@	Commercial at
125	27	'	Apostrophe
126	3D	=	Equals
127	22	"	Quotation marks

(Continued)

Table E-9. OSV$EBCDIC Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
128	C3	—	Unassigned
129	61	a	Lowercase a
130	62	b	Lowercase b
131	63	c	Lowercase c
132	64	d	Lowercase d
133	65	e	Lowercase e
134	66	f	Lowercase f
135	67	g	Lowercase g
136	68	h	Lowercase h
137	69	i	Lowercase i
138	C4	—	Unassigned
139	C5	—	Unassigned
140	C6	—	Unassigned
141	C7	—	Unassigned
142	C8	—	Unassigned
143	C9	—	Unassigned
144	CA	—	Unassigned
145	6A	j	Lowercase j
146	6B	k	Lowercase k
147	6C	l	Lowercase l
148	6D	m	Lowercase m
149	6E	n	Lowercase n
150	6F	o	Lowercase o
151	70	p	Lowercase p
152	71	q	Lowercase q
153	72	r	Lowercase r
154	CB	—	Unassigned
155	CC	—	Unassigned
156	CD	—	Unassigned
157	CE	—	Unassigned
158	CF	—	Unassigned
159	D0	—	Unassigned

(Continued)

Table E-9. OSV$EBCDIC Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
160	D1	—	Unassigned
161	7E	—	Unassigned
162	73	s	Lowercase s
163	74	t	Lowercase t
164	75	u	Lowercase u
165	76	v	Lowercase v
166	77	w	Lowercase w
167	78	x	Lowercase x
168	79	y	Lowercase y
169	7A	z	Lowercase z
170	D2	—	Unassigned
171	D3	—	Unassigned
172	D4	—	Unassigned
173	D5	—	Unassigned
174	D6	—	Unassigned
175	D7	—	Unassigned
176	D8	—	Unassigned
177	D9	—	Unassigned
178	DA	—	Unassigned
179	DB	—	Unassigned
180	DC	—	Unassigned
181	DD	—	Unassigned
182	DE	—	Unassigned
183	DF	—	Unassigned
184	E0	—	Unassigned
185	E1	—	Unassigned
186	E2	—	Unassigned
187	E3	—	Unassigned
188	E4	—	Unassigned
189	E5	—	Unassigned
190	E6	—	Unassigned
191	E7	—	Unassigned

(Continued)

Table E-9. OSV$EBCDIC Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
192	7B	{	Opening brace
193	41	A	Uppercase A
194	42	B	Uppercase B
195	43	C	Uppercase C
196	44	D	Uppercase D
197	45	E	Uppercase E
198	46	F	Uppercase F
199	47	G	Uppercase G
200	48	H	Uppercase H
201	49	I	Uppercase I
202	E8	—	Unassigned
203	E9	—	Unassigned
204	EA	—	Unassigned
205	EB	—	Unassigned
206	EC	—	Unassigned
207	ED	—	Unassigned
208	7D	}	Closing brace
209	4A	J	Uppercase J
210	4B	K	Uppercase K
211	4C	L	Uppercase L
212	4D	M	Uppercase M
213	4E	N	Uppercase N
214	4F	O	Uppercase O
215	50	P	Uppercase P
216	51	Q	Uppercase Q
217	52	R	Uppercase R
218	EE	—	Unassigned
219	EF	—	Unassigned
220	F0	—	Unassigned
221	F1	—	Unassigned
222	F2	—	Unassigned
223	F3	—	Unassigned

(Continued)

Table E-9. OSV$EBCDIC Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
224	5C	\	Reverse slant
225	9F	—	Unassigned
226	53	S	Uppercase S
227	54	T	Uppercase T
228	55	U	Uppercase U
229	56	V	Uppercase V
230	57	W	Uppercase W
231	58	X	Uppercase X
232	59	Y	Uppercase Y
233	5A	Z	Uppercase Z
234	F4	—	Unassigned
235	F5	—	Unassigned
236	F6	—	Unassigned
237	F7	—	Unassigned
238	F8	—	Unassigned
239	F9	—	Unassigned
240	30	0	Zero
241	31	1	One
242	32	2	Two
243	33	3	Three
244	34	4	Four
245	35	5	Five
246	36	6	Six
247	37	7	Seven
248	38	8	Eight
249	39	9	Nine
250	FA	—	Unassigned
251	FB	—	Unassigned
252	FC	—	Unassigned
253	FD	—	Unassigned
254	FE	—	Unassigned
255	FF	—	Unassigned

Table E-10. OSV$EBCDIC6_FOLDED Collating Sequence

The ASCII codes not listed here (ASCII codes 0 through 1F and 7F through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	20	SP	Space
01	2E	.	Period
02	3C	<	Less than
03	28	(	Opening parenthesis
04	2B	+	Plus
05	21	!	Exclamation point
06	26	&	Ampersand
07	24	$	Dollar sign
08	2A	*	Asterisk
09	29	)	Closing parenthesis
10	3B	;	Semicolon
11	5E,7E	^,~	Circumflex, tilde
12	2D	-	Hyphen
13	2F	/	Slant
14	2C	,	Comma
15	25	%	Percent sign
16	5F	_	Underline
17	3E	>	Greater than
18	3F	?	Question mark
19	3A	:	Colon
20	23	#	Number sign
21	40,60	@,`	Commercial at, grave accent
22	27	'	Apostrophe
23	3D	=	Equals

(Continued)

Table E-10. OSV$EBCDIC6_FOLDED Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	22	"	Quotation marks
25	5B,7B	[,{	Opening bracket, opening brace
26	41,61	A,a	Uppercase A, lowercase a
27	42,62	B,b	Uppercase B, lowercase b
28	43,63	C,c	Uppercase C, lowercase c
29	44,64	D,d	Uppercase D, lowercase d
30	45,65	E,e	Uppercase E, lowercase e
31	46,66	F,f	Uppercase F, lowercase f
32	47,67	G,g	Uppercase G, lowercase g
33	48,68	H,h	Uppercase H, lowercase h
34	49,69	I,i	Uppercase I, lowercase i
35	5D,7D	],}	Closing bracket, closing brace
36	4A,6A	J,j	Uppercase J, lowercase j
37	4B,6B	K,k	Uppercase K, lowercase k
38	4C,6C	L,l	Uppercase L, lowercase l
39	4D,6D	M,m	Uppercase M, lowercase m
40	4E,6E	N,n	Uppercase N, lowercase n
41	4F,6F	O,o	Uppercase O, lowercase o
42	50,70	P,p	Uppercase P, lowercase p
43	51,71	Q,q	Uppercase Q, lowercase q
44	52,72	R,r	Uppercase R, lowercase r
45	5C,7C	\,|	Reverse slant, vertical line
46	53,73	S,s	Uppercase S, lowercase s
47	54,74	T,t	Uppercase T, lowercase t
48	55,75	U,u	Uppercase U, lowercase u
49	56,76	V,v	Uppercase V, lowercase v
50	57,77	W,w	Uppercase W, lowercase w
51	58,78	X,x	Uppercase X, lowercase x

(Continued)

Table E-10. OSV$EBCDIC6_FOLDED Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	59,79	Y,y	Uppercase Y, lowercase y
53	5A,7A	Z,z	Uppercase Z, lowercase z
54	30	0	Zero
55	31	1	One
56	32	2	Two
57	33	3	Three
58	34	4	Four
59	35	5	Five
60	36	6	Six
61	37	7	Seven
62	38	8	Eight
63	39	9	Nine

Table E-11. OSV$EBCDIC6_STRICT Collating Sequence

The ASCII codes not listed here (ASCII codes 0 through 1F and 60 through FF hexadecimal) are ordered as equal to the space (ASCII code 20 hexadecimal).

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
00	20	SP	Space
01	2E	.	Period
02	3C	<	Less than
03	28	(	Opening parenthesis
04	2B	+	Plus
05	21	!	Exclamation point
06	26	&	Ampersand
07	24	$	Dollar sign
08	2A	*	Asterisk
09	29	)	Closing parenthesis
10	3B	;	Semicolon
11	5E	^	Circumflex
12	2D	-	Hyphen
13	2F	/	Slant
14	2C	,	Comma
15	25	%	Percent sign
16	5F	_	Underline
17	3E	>	Greater than
18	3F	?	Question mark
19	3A	:	Colon
20	23	#	Number sign
21	40	@	Commercial at
22	27	'	Apostrophe
23	3D	=	Equals

(Continued)

Table E-11. OSV$EBCDIC6_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
24	22	"	Quotation marks
25	5B	[	Opening bracket
26	41	A	Uppercase A
27	42	B	Uppercase B
28	43	C	Uppercase C
29	44	D	Uppercase D
30	45	E	Uppercase E
31	46	F	Uppercase F
32	47	G	Uppercase G
33	48	H	Uppercase H
34	49	I	Uppercase I
35	5D	]	Closing bracket
36	4A	J	Uppercase J
37	4B	K	Uppercase K
38	4C	L	Uppercase L
39	4D	M	Uppercase M
40	4E	N	Uppercase N
41	4F	O	Uppercase O
42	50	P	Uppercase P
43	51	Q	Uppercase Q
44	52	R	Uppercase R
45	5C	\	Reverse slant
46	53	S	Uppercase S
47	54	T	Uppercase T
48	55	U	Uppercase U
49	56	V	Uppercase V
50	57	W	Uppercase W
51	58	X	Uppercase X

(Continued)

Table E-11. OSV$EBCDIC6_STRICT Collating Sequence *(Continued)*

Collating Sequence Position	ASCII Code (Hexa-decimal)	Graphic or Mnemonic	Name or Meaning
52	59	Y	Uppercase Y
53	5A	Z	Uppercase Z
54	30	0	Zero
55	31	1	One
56	32	2	Two
57	33	3	Three
58	34	4	Four
59	35	5	Five
60	36	6	Six
61	37	7	Seven
62	38	8	Eight
63	39	9	Nine

FMU Conversion Rules, Storage Requirements, and Syntax Diagrams F

This appendix provides supplemental FMU information as follows:

- The rules FMU follows when converting one data type to another.
- The storage requirements for computational items.
- Statement syntax diagrams.

Data Type Conversion Between NOS/VE Files

FMU can convert any data type to any other data type. For example, it can convert data of type A (ASCII data) to data of type I (integer data) and vice versa.

The following matrix gives additional specific information on how FMU transforms one data type to another data type. The letter at the intersection of two data types refers to one of the rules listed after the matrix. A blank matrix entry indicates no additional information is given.

Input \ Output	A	B	F	G	H	I	J	L	N	P	Q	U	Y	Z
A	a	b	c	d	e	c	e	f	g	h	h	h	h	h
B	i			j					k					
F		l			l		l	m		l	l	l	n	n
G	o	p	q	o	p	o	p	r	q	p	p	p	p	p
H				s				m						
I				s				m						
J				t				u						
L	v		u	u	u	u	u		u	u	u	u	u	u
N	w	w	w	x	y	w	y	z	w	y	y	y	y	y
P				t				u						
Q				t				u						
U				t				u						
Y				t				u	A					
Z				t				u	A					

a. The source field is copied to the destination field, from left to right. If the source field is longer than the destination field, the source is truncated. If the source field is shorter than the destination field, the destination field is blank filled on the right.

b. The source is converted to F, if necessary, then to Z[,38], and finally, to the destination data type.

   The acceptable formats for numeric type A data are the same as those described for literals. Blanks are ignored.

   The floating-point conversion support is limited to values up through 9.9 x 10**34 (where ** represents exponentiation).

c. The acceptable formats for numeric type A data are the same as those described for literals. Blanks are ignored.

d. The source is converted to F, and then to the destination data type.

   The acceptable formats for numeric type A data is the same as those described concerning literals. Blanks are ignored.

e. The trailing or leading sign combined Hollerith support is limited to values up through 37 digits.

   The acceptable formats for numeric type A data are the same as those described concerning literals. Blanks are ignored.

   The floating-point conversion support is limited to values up through 9.9 x 10**34 (where ** represents exponentiation).

f. If the leftmost, nonblank character in the source field is T or t, the destination field is converted to the binary representation of TRUE. If the leftmost, nonblank character in the source field is F or f, the destination field is converted to the binary representation of FALSE. If the leftmost, nonblank character in the source field is a period (.), then the next character must be as previously specified.

g. The source is converted to integer (I) format, and then to the destination data type.

   The acceptable formats for numeric type A data are the same as those described concerning literals. Blanks are ignored.

h. The acceptable formats for numeric type A data are the same as those described concerning literals. Blanks are ignored.

The floating-point conversion support is limited to values up through 9.9 x 10**34 (where ** represents exponentiation).

i. The converted source is right-justified in the destination field, with zero fill to the left.

The numeric value of the B field is delivered to the destination A field as a base 10 number representation.

j. The source is converted to F, and then to the destination data type.

This conversion goes through a Z[,38] intermediate value transformation.

k. The source is converted to I, and then to the destination data type.

This conversion goes through a Z[,38] intermediate value transformation.

l. The floating-point conversion support is limited to values up through 9.9 x 10**34 (where ** represents exponentiation).

This conversion goes through a Z[,38] intermediate value transformation.

m. The integer value 0 is taken as FALSE, and nonzero as TRUE.

n. The floating-point conversion support is limited to values up through 9.9 x 10**34 (where ** represents exponentiation).

o. The source is converted to F, and then to the destination data type.

Blanks are ignored.

p. The source is converted to F, if necessary, then to the Z[,38], and finally, to the destination data type.

Blanks are ignored.

The floating-point conversion support is limited to values up through 9.9 x 10**34 (where ** represents exponentiation).

q. Blanks are ignored.

r. The integer value 0 is taken as FALSE, and nonzero as TRUE.

Blanks are ignored.

s. The source is converted to F, and then to the destination data type.

t. The source is converted to F, and then to the destination data type.

u. When converting to L, the integer value 0 is taken as FALSE, and nonzero as TRUE. When converting from L, the integer values 0 and 1 are used.

v. Either the word TRUE or FALSE is entered left-justified, blank-filled, in the destination field, as appropriate.

Rightmost truncation occurs if the desination field is too short for the value.

w. Blanks are ignored.

x. The source is converted to F, and then to the destination data type.

Blanks are ignored.

y. The source is converted to I, and then to the destination data type.

Blanks are ignored.

z. The integer value 0 is taken as FALSE, and nonzero as TRUE.

Blanks are ignored.

A. The source is converted to I, and then to the destination data type.

Storage Requirements for Computational Items

Table F-1 lists the NOS/VE storage requirements for computational items.

Table F-1. Storage Requirements for Computational Items

Unsigned Items				Signed Items		
A	**B**	**C**	**D**	**E**	**F**	**G**
1	4	8	1	5	8	1
2	7	8	1	8	8	1
3	10	16	2	11	16	2
4	14	16	2	15	16	2
5	17	24	3	18	24	3
6	20	24	3	21	24	3
7	24	24	3	25	32	4
8	27	32	4	26	32	4
9	30	32	4	31	32	4
10	34	40	5	35	40	5
11	37	40	5	38	40	5
12	40	40	5	41	48	6
13	44	48	6	45	48	6
14	47	48	6	48	48	6
15	50	56	7	51	56	7
16	54	56	7	55	56	7
17	57	64	8	58	64	8
18	60	64	8	61	64	8

Legend:

Column A	Digits in PICTURE Clause
Column B	Bits Required
Column C	Bits Rounded
Column D	Bytes Required
Column E	Bits + 1 Required
Column F	Bits + 1 Rounded
Column G	Bytes Required

Calculating Storage Requirements

The following equation can be used to determine the storage requirements for NOS/VE COMP items:

bytes = CEILING(INT(LOG2(n) + 2) /bits)

bytes	Number of bytes
CEILING (N)	Mathematical function that returns the smallest integer greater than or equal to N.
INT (N)	Mathematical function which truncates N to an integer.
LOG2 (N)	Logarithm to the base 2 function. LOG2 is found by the equation LOG2(N) = LN(N)/LN(2), where LN is the natural log function.
n	Largest number representable in the field.
bits	Number of bits per byte.

Examples

- In a NOS/VE COBOL program, a field defined as:

```
PIC S9(8)
```

has a storage requirement of 4 bytes:

CEILING(INT(LOG2(99999999) + 2) /8) = 4

FMU Statement Syntax Diagrams

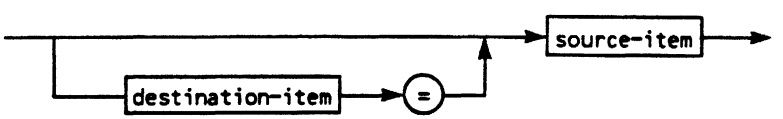
The syntax diagrams in this appendix can be used for a quick reference. They primarily apply to the CREATE_OUTPUT_RECORD (CREOR) statement-list syntax. To use these diagrams, remember the following notational rules:

Rounded boxes enclose actual syntax elements (keywords, literal values, and delimiters). Keyword abbreviations have been used. The following are examples:

Rectangles enclose names of syntax diagram sections. For example:

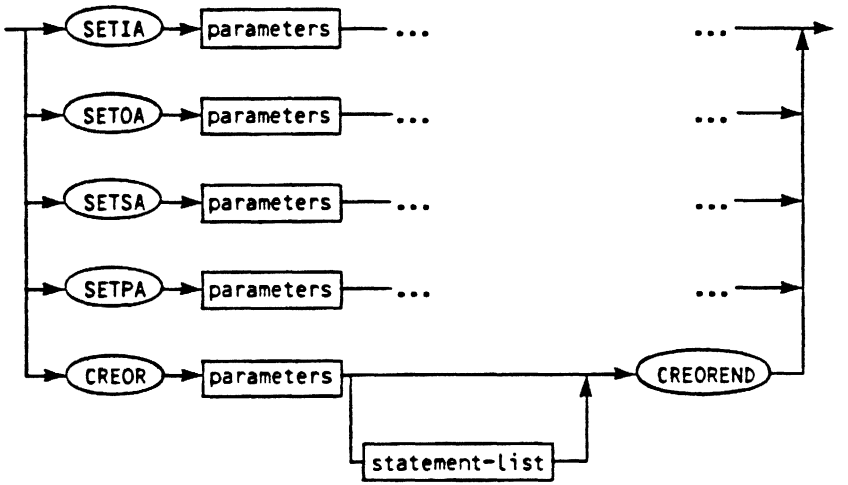
Arrows connecting the boxes show legal syntax of statements. For example:

assignment statement

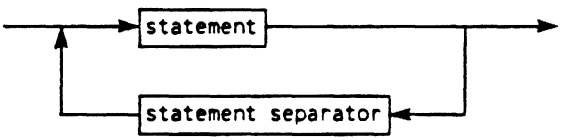


The diagram tells you that an assignment statement can either be a source-item alone or a source-item preceded by a destination-item and the '='. (Blanks can be inserted wherever there is a line.)

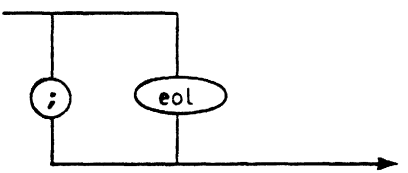
The following diagram shows the FMU directives and the general syntax. For full descriptions of the directives, refer to chapter 11. The main set of diagrams in this appendix show the CREATE_OUTPUT_RECORD directive statement-list syntax.



statement list

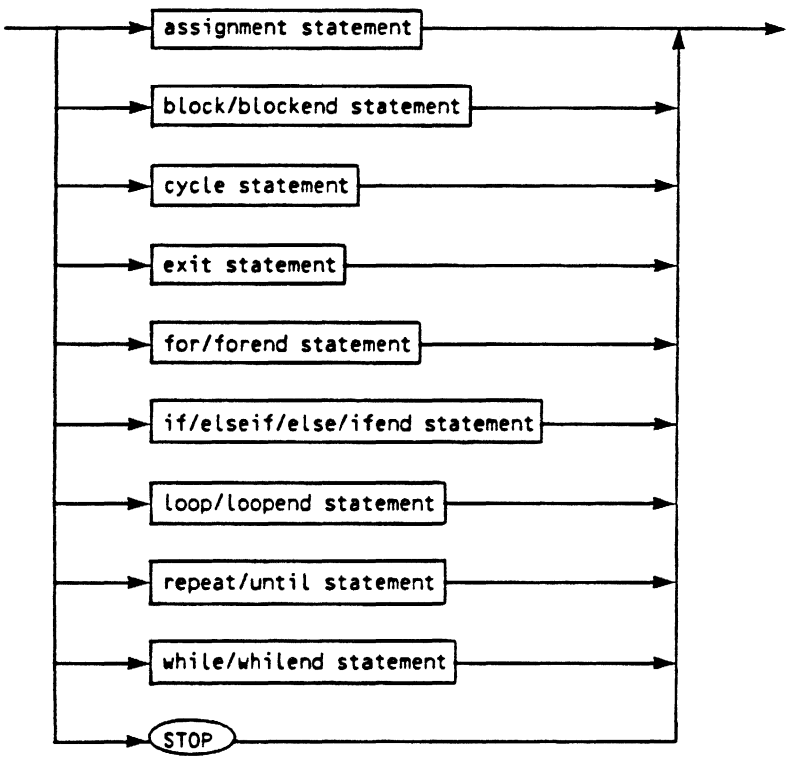


statement separator

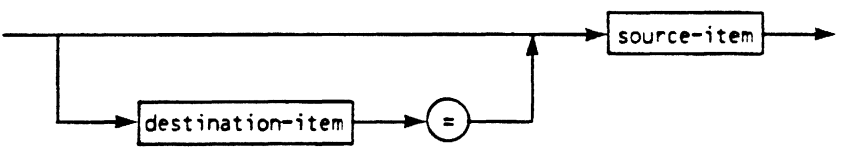


<eol> denotes end-of-line.

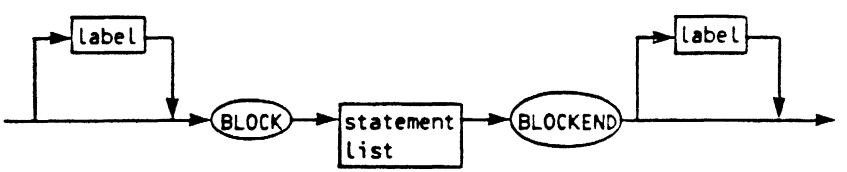
statement



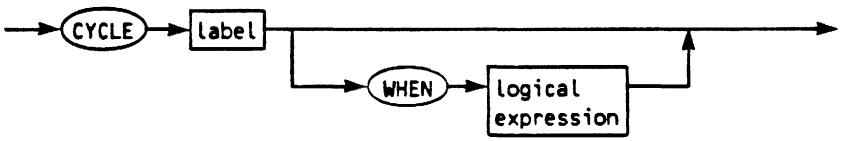
assignment statement



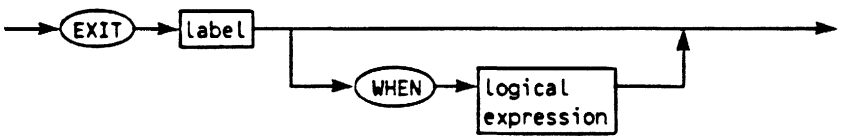
block/blockend statement



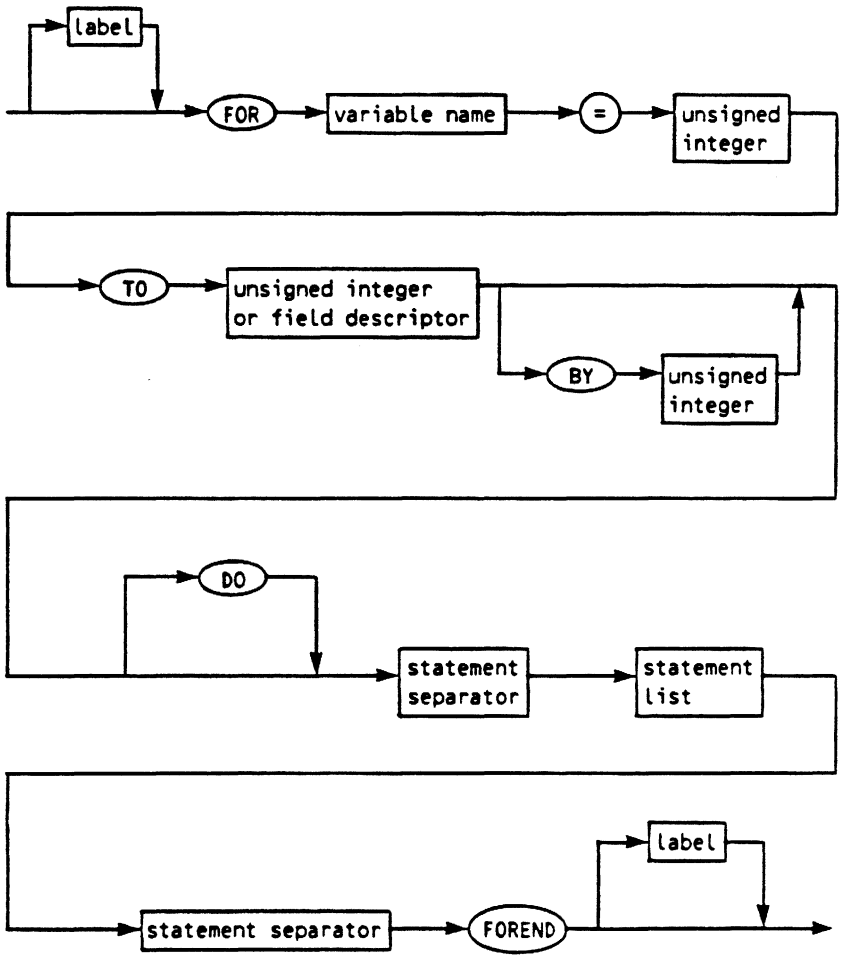
cycle statement



exit statement



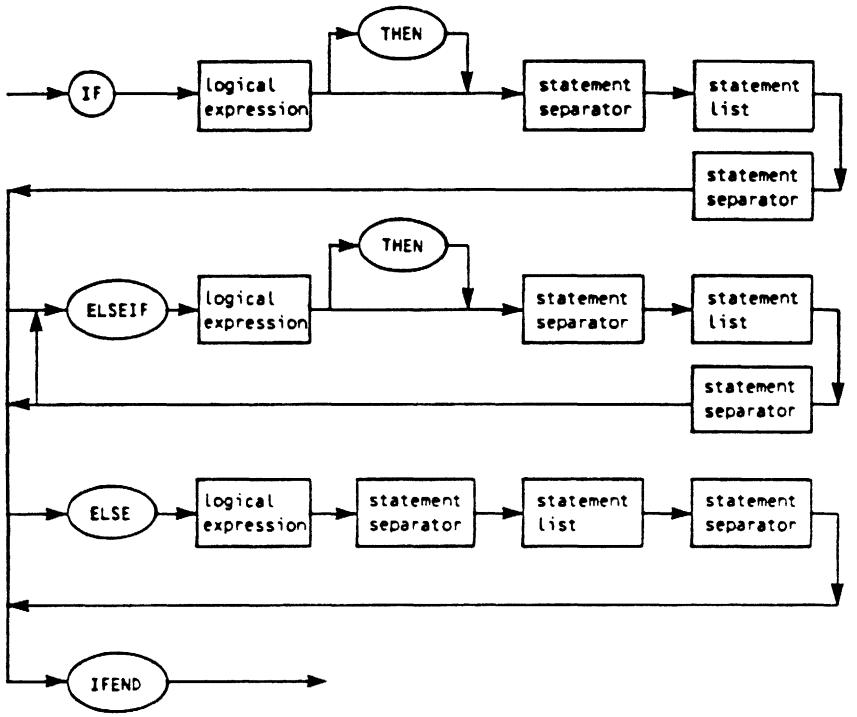
for/forend statement



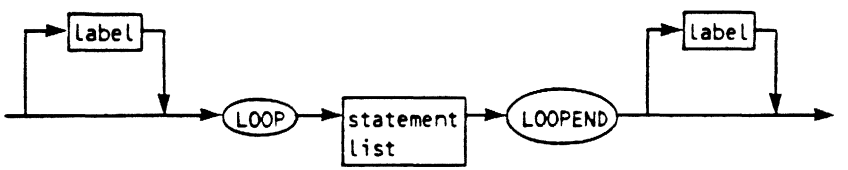
variable name: alphabetic character followed by zero or more alphanumeric characters.

unsigned integer: one or more numeric digits (0 through 9).

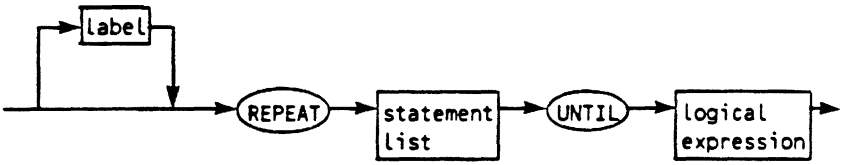
if/elseif/else/ifend statement



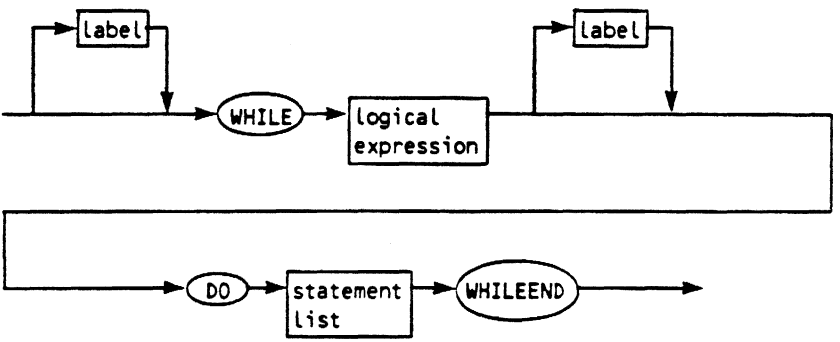
loop/loopend statement



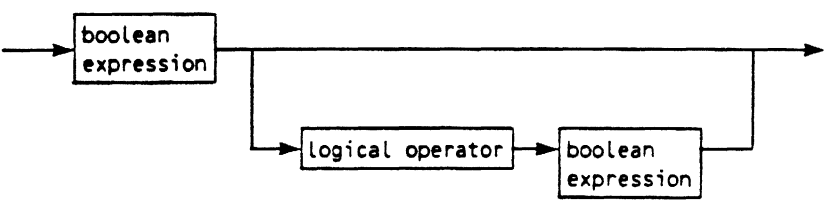
repeat/until statement



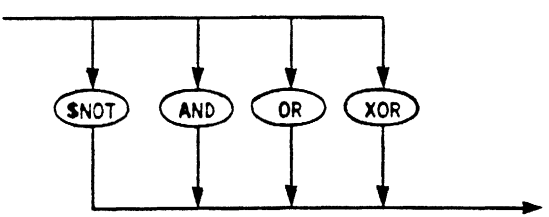
while/whilend statement



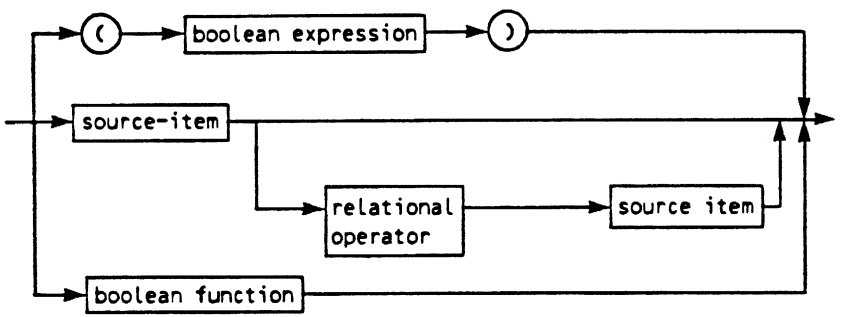
logical expression



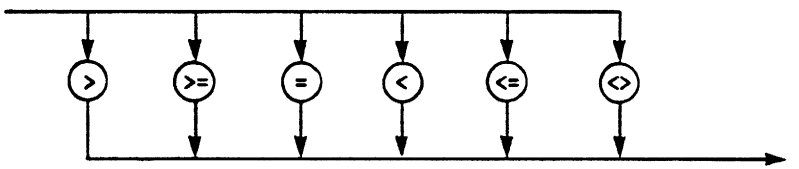
logical operator



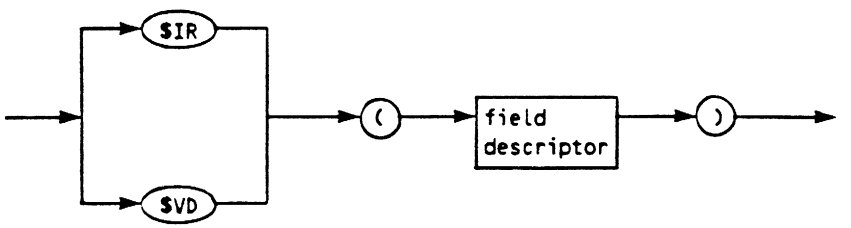
boolean expression



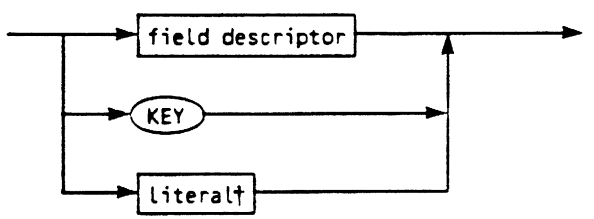
relational operator



boolean function

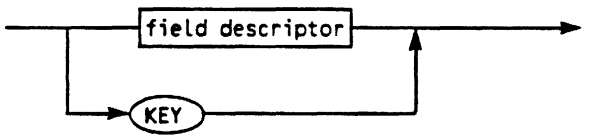


source-item

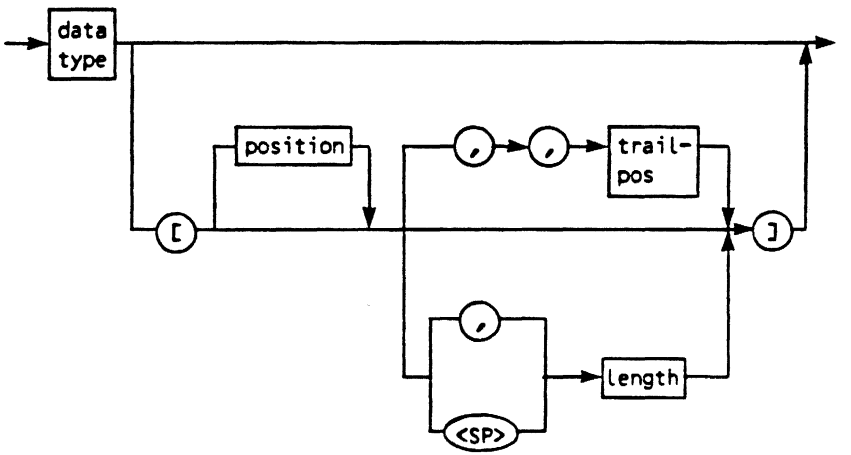


†character string enclosed in apostrophes.

destination-item

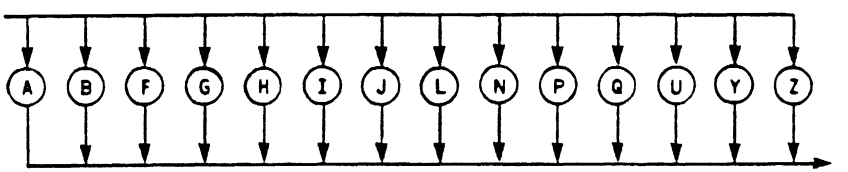


field descriptor

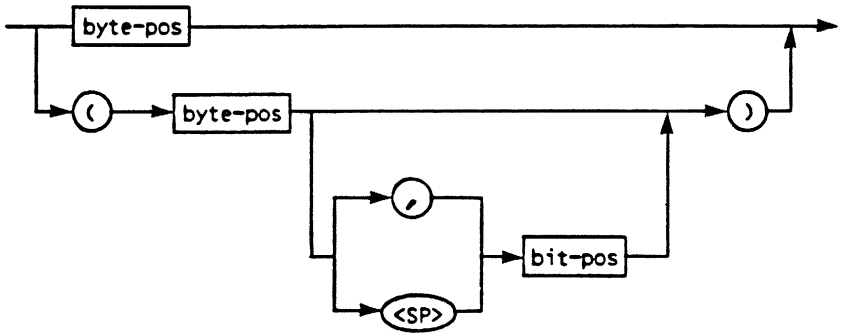


<SP> denotes a blank space.

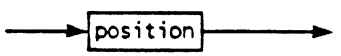
data type



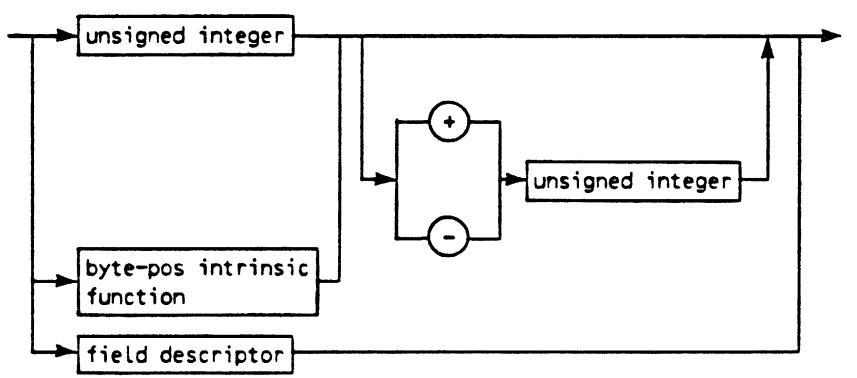
position



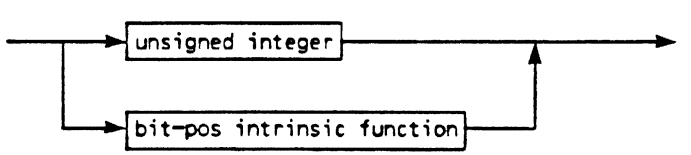
trail-pos



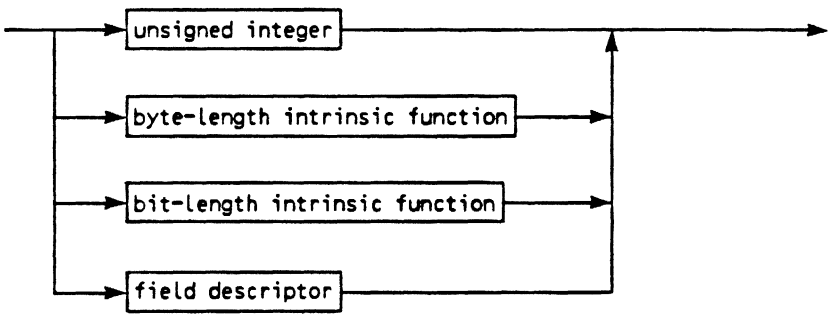
byte-pos



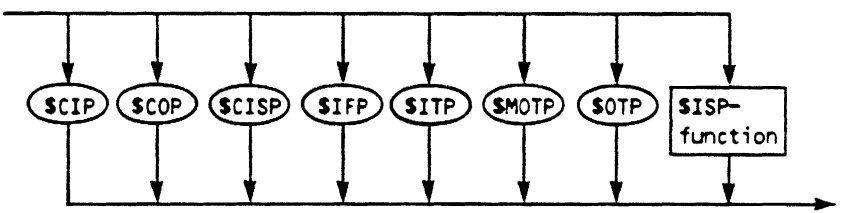
bit-pos



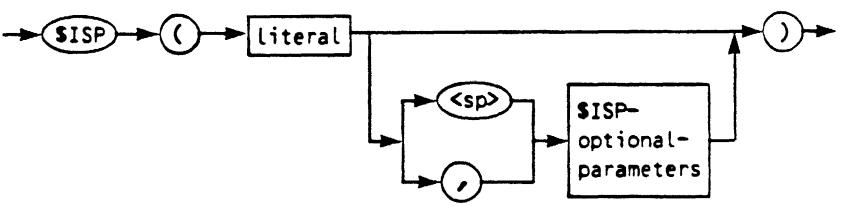
length



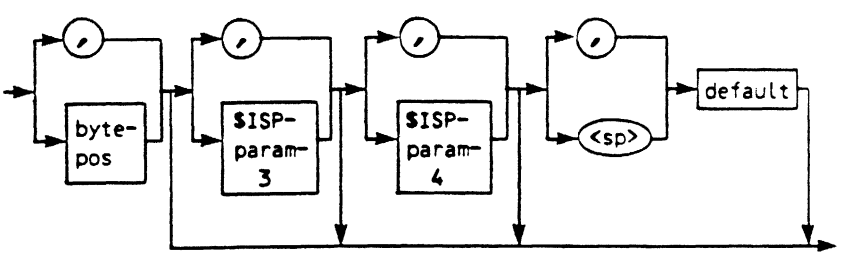
byte-pos intrinsic function



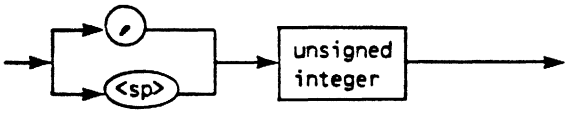
$ISP-function



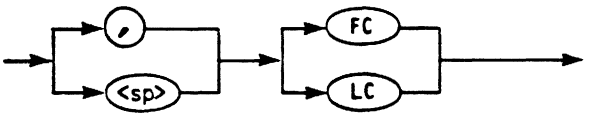
$ISP-optional-parameters



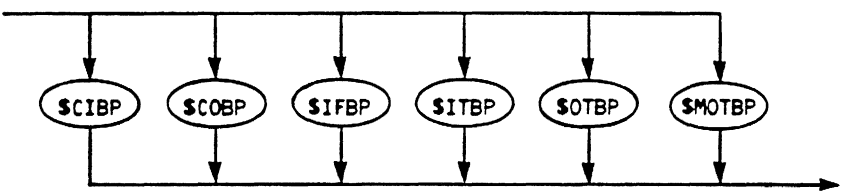
$ISP-param-3



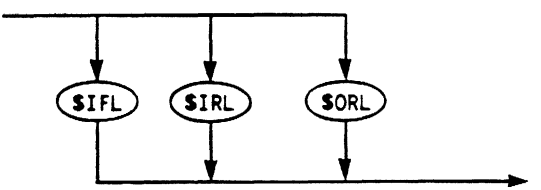
ISP-param-4



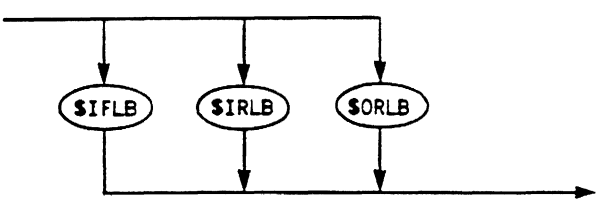
bit-pos intrinsic function



byte length intrinsic function



bit length intrinsic function



FMU Error Messages G

This appendix describes the messages returned by FMU processing. It first describes FMU error processing in general and then lists the messages FMU can write to the listing file.

About FMU Diagnostics

FMU issues three types of error diagnostics: compilation, execution, and system. The FMU system diagnostics are listed in the NOS/VE Diagnostic Messages manual (the FM messages, status codes 630000 through 639999).

Compilation diagnostics are produced when FMU reads the directive file and finds error conditions. The diagnostics are written to the list file. You can correlate them with the directive source listing, which also appears on the list file.

Execution diagnostics occur after the directives have been read. Execution diagnostics are written to the list file after the compilation diagnostics are written. These can be checked against the directive source listing.

System diagnostics are produced when an error is detected by a system routine used by FMU.

Error Message Communication

If any diagnostics are issued to the list file, the following status message is returned:

'Error details are on the LIST file.'

The list file is the file specified in the LIST (L) parameter of the FMU command. If you do not specify a file, $LIST is assumed. In batch mode, $LIST is associated with the OUTPUT file, which is printed. In interactive mode, $LIST is connected to the $NULL file and so, the messages sent to $LIST are discarded.

When an error occurs, the diagnostic is issued to the list file with the appropriate source line and column so that you can correlate the diagnostic list with the source listing produced during compilation. If no directives file was given, source line 0 and column 0 are used.

Error Handling

Usually, FMU terminates processing if a compilation error occurs during the directive syntax check.

The ERROR_DISPOSITION parameter on the FMU command determines whether FMU processing aborts if processing of an output file aborts. The ERROR_DISPOSITION parameter on the SET_OUTPUT_ATTRIBUTES directive for each output file determines whether generation of the output file aborts if an error occurs during output record formatting.

The CONVERSION_ERROR_DISPOSITION parameter on the SET_OUTPUT_ATTRIBUTES directive determines whether recovery is attempted if a source field contains unrecognizable data. If the CONVERSION_ERROR_DISPOSITION parameter specifies RECOVER, a data error during assignment causes a default value to be used for the source field value; no diagnostic message is issued. The default value used depends on the data type of the source item as follows:

Source Item	Default Value
Data type A	Spaces
Data type L	Logical false (.FALSE.)
All other data types	Numeric zero in the appropriate format

If the CONVERSION_ERROR_DISPOSITION parameter on the SET_OUTPUT_ATTRIBUTES directive specifies ABORT (the default value), a data error during assignment discards the current output record. If an output file error occurs other than during assignment, the current output record is discarded regardless of the CONVERSION_ERROR_DISPOSITION parameter specification.

Discarded records are written to the exception records file if one is specified by the EXCEPTION_RECORDS_FILE parameter of the SET_OUTPUT_ATTRIBUTES directive.

Diagnostic Templates

The diagnostics consist of a basic template on which the whole diagnostic is built. Messages specific to an error condition are inserted or appended to the templates to give the full message.

In the diagnostics listed in this manual, these messages are indicated by text enclosed in braces ({text}). Inserts, however, are not limited to those places. At times when insertion spaces in the template are filled, inserts might be concatenated to the end of the diagnostic being built.

Special conditions can produce a hierarchical diagnostic. For example, if one of the system routines that FMU calls fails, then the returned status is expanded and returned to the template:

'Abnormal status detected:'

This text might be appended to yet another template which further defines the error condition.

Error Severities

These are the severities of errors that are shown on the source listing:

Nonstandard (NS)

Trivial if ERROR_DISPOSITION=NO_ABORT and fatal if ERROR_DISPOSITION=ABORT, for the ERROR_DISPOSITION specification given on the SET_OUTPUT_ATTRIBUTES directive.

Dependent (D)

Usage is product or machine dependent.

Trivial (T)

Error is not major; processing continues.

Warning (W)

Problem is uncorrected; error is ignored. User is warned.

Fatal (F)

Error causes processing to stop.

Catastrophic (C)

Internal product or system error; processing stops.

Messages Listing

The rest of this appendix lists messages that FMU can write to the listing file. The messages are listed in alphabetical order.

Messages that begin with an insertion item (such as a file name) are placed at the end of the messages listing.

A BLOCK/BLOCKEND block cannot be cycled.

Description The label on the CYCLE statement matches the label on a BLOCK statement.

User Action Correct the label or remove the CYCLE statement.

A buffer cannot be acquired.

Description FMU could not acquire another segment.

User Action Either use fewer output files or ask site personnel for help.

A character appears more than once in the collating sequence specification.

Description An ASCII character has been specified more than once in the collating sequence specification.

User Action Check the collating sequence specification for duplicate characters. In particular, check that no ranges overlap.

A collating sequence specification must contain at least one character.

Description An empty collating sequence was specified. (The enclosing parentheses contained no values.)

User Action Specify values for at least one step in the enclosing parentheses. To assign all characters to the same position in the collating sequence, specify a range that includes all characters (0..255).

A collating sequence sublist must contain at least one character.

Description A set of enclosing parentheses within the collating sequence specification contains no values.

User Action Each position specified must contain at least one character. Remove the extra pair of parentheses or specify at least one character in the parentheses.

A directive was specified more than once per file.

Description More than one directive of the same type (such as SET_OUTPUT_ATTRIBUTES) specified the same file.

User Action Remove one of the duplicate directives or change the name specified by one of the FILE parameters, whichever is appropriate.

A duplicate of the specified CREATE_OUTPUT_RECORD statement list cannot be used for this file, as this file has no keys defined and yet the specified statement list uses KEY.

Description The CREATE_OUTPUT_RECORD directive specifies the DUPLICATE_SPECIFICATION parameter. The CREATE_OUTPUT_RECORD specification to be duplicated specifies the word KEY, referencing the primary key of a record. However, the file to use the duplicated specification does not have a primary key defined by its file attributes and so the duplicated statement list cannot be used.

User Action First, check whether the correct file name is specified on the DUPLICATE_SPECIFICATION parameter. Next, check whether a primary key should be defined for the output file. (If so, use the SET_FILE_ATTRIBUTES command to specify the KEY_POSITION and KEY_LENGTH attribute values.) If necessary, remove the DUPLICATE_SPECIFICATION parameter and explicitly specify the output record formatting for the file.

A label is not permitted on this statement.

Description A label is permitted only on a structured statement (BLOCK, FOR, IF, LOOP, REPEAT, or WHILE).

User Action Remove the label.

A literal that starts or ends a range must contain only one character.

Description A string specifying the lower or upper bound of a range contains more than one character.

User Action Check that the correct range is being specified. The character beginning and ending the range should be enclosed in apostrophes ('A'..'Z') or specified as its integer code (65..90).

A negative number cannot be moved to an unassigned binary field.

Description An input record provided a negative value as the value to be assigned to a field of type B. If ERROR_DISPOSITION=NO_ABORT is specified for the file, this error is nonfatal; otherwise, it is a fatal error.

User Action Review the CREATE_OUTPUT_RECORD directive for the file to ensure that the correct data formatting has been specified. If appropriate, change the data type from B to a data type that allows negative values.

Abnormal STATUS detected : {abnormal status message}

Description FMU returned an abnormal completion status as described in the message.

User Action Determine the reason for the failure using the specific information in the message.

An ASCII ordinal must be between 0 and 255, inclusive.

Description The collating sequence specification contains a number that is not an ASCII character code (0 through 255 decimal).

User Action Correct the number that is out of range. Each character can be represented by its graphic enclosed in apostrophes or by its ordinal within the ASCII character set (decimal integer).

An unrecognized character was detected.

Description A non-ASCII character is not valid in the directive file.

User Action Check that the correct file was specified on the DIRECTIVES parameter. If the file specification is correct, correct the character at the specified line and column.

An unrecognized CONVERSION_ERROR_DISPOSITION of {keyword} was detected. Acceptable values are ABORT or RECOVER.

Description The value specified for the CONVERSION_ERROR_DISPOSITION parameter is incorrect.

User Action Specify ABORT or A (the default value) or RECOVER or R. ABORT specifies that a data error terminates reformatting of the output record and discards the record; RECOVER specifies that the default values be used for invalid data and reformatting continues.

An unrecognized ERROR_DISPOSITION of {keyword} was detected. Acceptable values are ABORT or NO_ABORT.

Description The value specified for the ERROR_DISPOSITION parameter is incorrect.

User Action Specify ABORT or A (the default value) or NO_ABORT or NA.

An unrecognized file unit type was detected. Acceptable options are RECORDS or PARTITIONS.

Description The second value in the value set for the MAXIMUM_FILE_UNITS (MFU) parameter must be either RECORDS (R) or PARTITIONS (P). RECORDS specifies that the first value in the value set is the maximum number of records; PARTITIONS specifies that the first value is the maximum number of partitions.

User Action Correct the value set for the MAXIMUM_FILE_UNITS parameter.

An unrecognized MACHINE_FORMAT of {keyword} was detected. Acceptable values are C170, C180, C7600, IBM, VAX, or VAXG.

Description The value specified for the MACHINE_FORMAT (or MF) parameter is incorrect.

User Action For the input file, specify the machine on which the file was written. For the output file, specify the machine on which the file will be used.

An unrecognized PARTITION_DISPOSITION was detected. Acceptable values are INCLUDE_PARTITIONS or EXCLUDE_PARTITIONS.

Description The value specified for the PARTITION_DISPOSITION (or PD) parameter is incorrect.

User Action Specify whether partition boundaries are to be included (IP) (the default value) or excluded (EP) for the output file.

An unrecognized RECORD_PRESET_VALUE of {keyword} was detected. Acceptable values are NO_PRESET, CHARACTER_BLANKS, CHARACTER_ZEROES, BINARY_ZEROES, or INPUT_RECORD.

Description The value specified for the RECORD_PRESET_VALUE (RPV) parameter on a CREATE_OUTPUT_RECORD directive is incorrect.

User Action Specify a valid value or omit the parameter. (The default value is NO_PRESET.)

An unrecognized skip direction was detected. Acceptable values are FORWARDS or BACKWARDS.

Description The value specified as the skip direction in the STARTING_FILE_POSITION (or SFP) value set is incorrect.

User Action Specify the third value in the value set as FORWARD (F) (the default value) or BACKWARD (B).

An unrecognized PRINT_FORMAT of {value} was detected. Acceptable values are 1, 2, 3, or DUMP.

Description The value specified for the PRINT_FORMAT (or PF) parameter is incorrect.

User Action Specify the line spacing as 1, 2, or 3 or specify dump format (DUMP). The default value is 1.

Attributes to define the key were not provided for file {name}.

Description The CREATE_OUTPUT RECORD directive for the file referenced a KEY field but the attributes that define the key were not set for the file.

User Action For a new NOS/VE output file, enter a SET_FILE_ATTRIBUTES command for the file before the FMU command. The SET_FILE_ATTRIBUTES command must specify at least the file name, the file organization, the maximum record length, and the key position and length.

Conflicting file usage -- file {file name} is used both as {parameter} and {parameter}.

Description The same file was specified for two different purposes in the FMU run.

User Action Check that the correct file names were specified. Change one or both file names so that different files are used.

CYCLE must be used inside a LOOP, FOR, WHILE, or REPEAT block.

Description The CYCLE statement is not in a statement list that repeats.

User Action Check the logic of the CREATE_OUTPUT_RECORD statement list. If appropriate, remove the CYCLE statement.

Directive file line cannot be acquired.

Description FMU cannot get the next line from the directive file.

User Action Check that the correct directives file was specified. Ask for help from site personnel if necessary.

Division by zero attempted.

Description Evaluation of an arithmetic expression would require division by a zero value, which is undefined. If the divisor is a field descriptor, this error occurs when the value in the specified field is zero.

User Action Change the arithmetic expression so that it does not attempt a division by zero. Do not specify a field that could contain a zero value as the divisor in an expression.

DUPLICATE_SPECIFICATION and FILE parameters specify same name.

Description The file specified on the FILE (or F) parameter of the directive is the same as the file specified on the DUPLICATE_SPECIFICATION (DS) parameter of the same directive.

User Action Correct the file name on the F or DS parameter, or omit the DS parameter and specify the other parameters explicitly on the directive.

Duplicate {name} parameter in {name} directive.

Description A directive cannot have duplicate parameters.

User Action Remove the redundant or conflicting parameters.

Exception record file {file name} cannot be opened.

Description FMU could not open and write to the file specified as the exception record file.

User Action Check that the correct file is specified on the EXCEPTION_RECORDS_FILE parameter. The file name should be in the $LOCAL catalog. Attaching a permanent file puts its name in the $LOCAL catalog.

EXIT must be used inside a BLOCK, LOOP, FOR, WHILE, or REPEAT block.

Description The EXIT statement is not in the statement list of a structured statement.

User Action Check the logic of the CREATE_OUTPUT_RECORD statement list. If appropriate, remove the EXIT statement.

Field length of SEQUENCE_FIELD parameter is zero or not an integer constant.

Description The second value in the field descriptor specified for the SEQUENCE_FIELD (or SF) parameter must be a nonzero integer constant.

User Action Correct the descriptor for the sequence field.

Field length too big for data type. Maximum length is {number}.

Description The length specified in the field descriptor exceeds the maximum length for the data type for the machine format of the file.

User Action Correct the length or data type of the field descriptor.

Field position of SEQUENCE_FIELD parameter is not integer constant.

Description The first value in the field descriptor specified for the SEQUENCE_FIELD (or SF) parameter must be an integer constant.

User Action Correct the field descriptor.

File attributes for {file} cannot be acquired.

Description FMU cannot acquire the file attribute values of the file.

User Action Check that the correct file was specified. If necessary, ask site personnel for help.

File {name} cannot be located to duplicate its specification.

Description The file specified by the DUPLICATE_SPECIFICATION (or DS) parameter is not specified by the FILE (or F) parameter of a preceding SET_OUTPUT_ATTRIBUTES directive.

User Action Check that the correct file names were specified on the DS parameter and on the preceding SET_OUTPUT_ATTRIBUTES directives. Then, either move the SET_OUTPUT_ATTRIBUTES directive for the file before this directive or change this directive so that it omits the DS parameter and explicitly specifies the other parameter values.

File {name} cannot be opened.

Description FMU cannot open the file.

User Action Check that the correct file was specified. If necessary, ask site personnel for help.

File does not have a collation table.

Description The CONDITION_COLLATING_SEQUENCE parameter specifies that the collation table stored with the input file (INPUT_FILE) or output file (OUTPUT_FILE) is to be used. However, no collation table is stored with the file because it is not an indexed-sequential file with a collated primary key.

User Action Check that the correct file was specified as the input file or output file. Specify a collation table on the CONDITION_COLLATING_SEQUENCE parameter.

File {file name} has already been specified as an exception record file.

Description The file has already been specified as the exception records file for another output file. (It was specified as the value for the EXCEPTION_RECORDS_FILE parameter on another SET_OUTPUT_ATTRIBUTES directive.)

User Action Check that the correct file name was specified. Specify another file on one of the EXCEPTION_RECORDS_FILE parameters.

File {file name} has been prematurely closed.

Description The file has been closed because FMU found an error and the ERROR_DISPOSITION specification is ABORT (the default).

User Action See the other diagnostics for the reason for the file closing.

File {name} is not defined in a SET_OUTPUT_ATTRIBUTES directive.

Description The file name specified on the directive is not specified on the FILE (or F) parameter of a preceding SET_OUTPUT_ATTRIBUTES directive.

User Action Supply or relocate the SETOA directive for the file.

FILE parameter is missing from {directive name} directive.

Description The required FILE (or F) parameter is missing from the directive.

User Action Add the FILE parameter specifying the file name.

FMU malfunction:

Description FMU has detected an error in its own processing.

User Action Follow site-defined procedures for reporting software problems.

FOR loop step value cannot be zero.

Description The FOR statement specifies a step value of 0. The FOR statement increment can be a positive or negative integer, but it cannot be zero. The default value is 1.

User Action Correct or remove the step value.

Found a DUPLICATE_SPECIFICATION parameter duplicating an unspecified directive.

Description The DUPLICATE_SPECIFICATION cannot be used if the directive has not been already defined for the specified file.

User Action Specify the directive before duplicating its specification.

Found DUPLICATE_SPECIFICATION parameter coexisting with parameters other than FILE.

Description If the DUPLICATE_SPECIFICATION (DS) parameter is specified on a directive, the only other parameter that can also be specified on the directive is the FILE parameter. Specification of any additional parameters is invalid.

User Action Remove either the DS parameter or the parameters other than the DS and FILE parameters.

Found {item} in {language element} where expecting {item}.

Description A syntax error in a language element of the directive file was detected.

User Action Correct language element. Minimum requirement is to supply one of the expected items in place of the found item. More might be required, however, because this error causes FMU to skip all characters out to the next semicolon or end-of-line.

Found length specification of one for output field of data type Y, Z, or G.

Description The length specification for data types Y, Z, and G must be greater than one byte.

User Action Make length specification greater than one.

Found MACHINE_FORMAT of file specified by DUPLICATE_SPECIFICATION parameter to be different from MACHINE_FORMAT of file specified by FILE parameter.

Description The MACHINE_FORMAT parameter value for the file specified by the DUPLICATE_SPECIFICATION (or DS) parameter must be the same as for the file specified by the FILE (or F) parameter.

User Action Specify the same MACHINE_FORMAT value for both files.

Found more than two values in MAXIMUM_FILE_UNITS parameter.

Description The value set for the MAXIMUM_FILE_UNITS (or MFU) parameter can contain only two values: the unit type and the number of units.

User Action Correct the MFU value set.

Given file attributes of INTERNAL_CODE = {option}, data type {type} is not valid for {file name} file. Acceptable values are {types}.

Description The internal code file attribute indicates that the specified data type is invalid. If not explicitly specified, the internal code is derived from the MACHINE_FORMAT specification for the file. The internal code for C170 MACHINE_FORMAT is D64, which disallows use of the P (packed decimal) data type.

User Action Change the data type to one of the acceptable types.

In order to use SET_PRINT_ATTRIBUTES for this file, the maximum record length must be at least {number}.

Description To format a file for printing, FMU requires that the maximum record length attribute of the file be at least the value specified in the message. FMU determines the value in the message using the value of the page width attribute.

User Action Either remove the SET_PRINT_ATTRIBUTES directive specifying the file so FMU does not format the file for printing or increase the maximum record length value to the value specified in the message.

$INPUT_STRING_POS cannot be used on a C170 file containing ASCII data.

Description The string search function, $INPUT_STRING_POS ($ISP) cannot be used when the file is a NOS or NOS/BE ASCII (uppercase and lowercase) text file.

User Action Remove the $INPUT_STRING_POS function.

$INPUT_STRING_POS could not find the specified search string.

Description The string search function, $INPUT_STRING_POS ($ISP) did not find the specified string in the search.

User Action Check that the correct values are specified for the function. If appropriate, specify a default value to be used when the string is not found in an input record.

Insufficient room in the destination field for the final 12-bit 6/12 ASCII character.

Description The destination field specified on the assignment statement is too short for the converted data.

User Action Lengthen the destination field.

Integer constant is too large.

Description A specified integer constant is greater than $2^{32} - 1$.

User Action Ensure that the constant ends with a valid delimiter such as a space. Decrease the constant if necessary.

Integer value is outside permissible range of {number} to {number} for this context.

Description Integer value is either too large or too small.

User Action Correct the integer value.

INTERNAL_CODE attribute value invalid for C170 file.

Description The IC attribute value is invalid.

User Action Correct the IC attribute value.

INTERNAL_CODE attribute value invalid for C180 file.

Description The INTERNAL_CODE (or IC) attribute value is invalid.

User Action Specify ASCII as the INTERNAL_CODE file attribute value.

Length is bad for data type {type}. Acceptable values are {values}.

Description The length specified in the field descriptor is invalid for the indicated data type.

User Action Either correct the data type or the length.

Literal exceeds {number} characters.

Description The literal is longer than the maximum length for a literal of that data type.

User Action Check length of literal or change data types to accommodate the literal.

Literal is not terminated by an apostrophe.

Description A literal must be terminated by an apostrophe.

User Action Terminate the literal with an apostrophe, and check for proper line continuation where appropriate. Or, check whether an apostrophe is to be embedded in the literal.

MAXIMUM_RECORD_LENGTH for {file name} exceeds {number} characters.

Description FMU cannot handle records this large.

User Action Decrease the MAXIMUM_RECORD_LENGTH (or MAXRL) attribute value for the file.

Name exceeds {number} characters.

Description A name specified on a directive is too long.

User Action Shorten the name.

No directives were found in the DIRECTIVES file.

Description The FMU command specified a directives file, but the file contains no FMU directives.

User Action Correct the file, or specify the correct DIRECTIVES file name.

No keys for the indexed_sequential output file were defined.

Description A file was designated as an indexed-sequential file, but no key was defined. This error is fatal.

User Action Check that a primary key is defined by the NOS/VE SET_FILE_ATTRIBUTE command or the NOS or NOS/BE FILE command.

No records were read from the input file.

Description FMU did not read any records from the input file.

User Action Check that the input file is local. (For NOS/VE files, this means the file must be in the $LOCAL catalog; an ATTACH_FILE command enters a permanent file name in the $LOCAL catalog.)

Next, check that the file is not positioned at its end when FMU reads it. To ensure that the file is at its beginning, rewind the file before the FMU command.

Finally, check whether the file is empty.

No SET_OUTPUT_ATTRIBUTES directive was specified in the DIRECTIVES file.

Description The SET_OUTPUT_ATTRIBUTES directive is required when a directive file is used. This error is fatal.

User Action Specify the SETOA directive for the output file.

Null literal detected.

Description A literal contained no characters.

User Action Place characters in the literal. To specify a literal containing only an apostrophe, specify "". To specify a literal containing only a space, specify ' '.

Position of sequence field is outside range of 1 to {number}.

Description The position of the sequence field must be within the maximum record length of the file.

User Action Increase the MAXIMUM_RECORD_LENGTH (or MRL) attribute value or decrease the sequence field position value.

Presence of DUPLICATE_SPECIFICATION parameter in directive conflicts with presence of statement list.

Description When the DUPLICATE_SPECIFICATION (or DS) parameter is specified by the CREATE_OUTPUT_RECORD directive, only the FILE parameter and the terminator CREATE_OUTPUT_RECORD_END (or CREOREND) are permitted.

User Action Specify only the allowed parameters and terminator, or remove the DS parameter.

PRINT_TITLE literal exceeds {number} characters.

Description The PRINT_TITLE literal exceeds the maximum number of characters allowed.

User Action Shorten the print title.

Redefinition of SET_INPUT_ATTRIBUTES directive detected.

Description The SET_INPUT_ATTRIBUTES can be used only once in the directive file.

User Action Specify only one SETIA directive.

SEQUENCE_FIELD parameter absent from SET_SEQUENCE_ATTRIBUTES directive.

Description The SEQUENCE_FIELD parameter is required on this directive.

User Action Specify a sequence field descriptor on the directive.

SEQUENCE_NUMBER_INCREMENT parameter value is zero.

Description The value specified by the SEQUENCE_NUMBER_INCREMENT parameter must be greater than zero.

User Action Correct the SNI value.

SET_PRINT_ATTRIBUTES directive cannot be used on a C170 file containing ASCII data.

Description A CYBER 170 ASCII file (uppercase and lowercase text) cannot be formatted for printing.

User Action Check that the correct file name is specified. If so, remove the SET_PRINT_ATTRIBUTES directive.

SET_PRINT_ATTRIBUTES directive specified for file with record type not V.

Description A file specified on a SET_PRINT_ATTRIBUTES (SETPA) directive must have CDC variable (V) records as its RECORD_TYPE (RT) attribute value.

User Action Check that the correct file is specified on the directive. Specify the V record type for the file before it is created.

SET_PRINT_ATTRIBUTES directive specified for non-sequential file.

Description The SET_PRINT_ATTRIBUTES directive specifies a file to be formatted for printing. A file to be printed must have sequential file organization.

User Action Check that the correct file name was specified on the directive. Change the SET_FILE_ATTRIBUTES command for the file so that it specifies sequential file organization. If changing the file organization is not appropriate, delete the SET_PRINT_ATTRIBUTES directive for the file.

Specified INPUT file cannot be located.

Description No file having the name specified on the FILE parameter exists in the $LOCAL file catalog.

User Action Attach the file before executing the FMU command.

Statement list absent from the CREATE_OUTPUT_RECORD directive.

Description FMU could not find a statement list in the CREOR directive.

User Action Specify the DUPLICATE_SPECIFICATION (or DS) parameter, or add a statement list to the CREOR directive. The statements must precede the CREATE_OUTPUT_RECORD_END (or CREOREND) terminator for the directive.

The bit offset part of the field's position is out of range.

Description The bit index is not within the permissible range. The range is 1 through 8 for a NOS/VE file and 1 through 6 for a NOS or NOS/BE file.

User Action Check data conversion operations.

The conversion cannot proceed as specified - possibly a field width problem or unsupported conversion requested.

Description The transformation could not proceed for some reason.

User Action Examine field widths or the desired conversion.

The C170 file {file name} cannot be accessed because FMU is not running within the CREATE_INTERSTATE_CONNECTION utility.

Description To access a file on the NOS or NOS/BE side of the system, FMU must be executed in a CREATE_INTERSTATE_CONNECTION (CREIC) utility session. The CREATE_INTERSTATE_CONNECTION command must precede the FMU command.

User Action Add a CREATE_INTERSTATE_CONNECTION command before the FMU command and a QUIT command after the FMU command.

The destination field was too small to contain the converted source.

Description The output field was too small for the value. This error is trivial if ERROR_DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Check the data values. Enlarge the length of the destination field if necessary.

The field exceeds input record bounds. The input record's length is {length}.

Description The field exceeds the length of the input record as defined by the maximum record length attribute specified by the SET_FILE_ATTRIBUTE or the NOS or NOS/BE FILE command. This error is trivial if ERROR_DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Correct the length on the field descriptor.

The field's length is not compatible with the data type specified. The length specified is {number}.

Description Certain data types have restrictions on the lengths that the field can have. This error is trivial if ERROR_ DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Change the data type or length.

The field's position is out of range.

Description The field position is outside of the range specified for the field. This error is trivial if ERROR_ DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Check the assignment statements.

The label does not match the one specified at the beginning of the block.

Description The trailing label does not match the preceding label on the statement.

User Action Either remove the trailing label or correct it to match the label at the beginning of the structured statement.

The record associated with the input file record [number {record number} <or> key {key value}] on FILE {file name}, was discarded.

Description An error was found in record processing, and the record was discarded. The record number appears in the insert for a sequential file; the key value appears for a keyed file. This error is trivial if ERROR_ DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Examine other error messages on the listing for the run.

The sequence number for FILE {file name} had to be reinitialized at the SEQUENCE_NUMBER_PRESET value.

Description The sequence number value became too large upon incrementing. Sequencing continued, beginning with the SEQUENCE_NUMBER_PRESET value. This error is trivial if ERROR_DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Define a larger sequence field.

The SEQUENCE_NUMBER_PRESET value for FILE {file name} will not fit into the SEQUENCE_FIELD.

Description The value specified as the initial sequence number is too large for the sequence field. This error is fatal.

User Action Decrease the initial sequence number or increase the sequence field length.

The source field contains data which is incompatible with the data type specified.

Description The type of data in the input field is not compatible with the specified data type. This error is trivial if ERROR_DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Change either the data type or the position and length of the field.

The source is indefinite.

Description The value of the input data is indefinite. This error is trivial if ERROR_DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Determine whether the field descriptor was correctly specified.

The source is infinite.

Description The input value is infinite. This error is trivial if ERROR_DISPOSITION=NO_ABORT is specified; otherwise, the error is fatal.

User Action Determine whether the field descriptor was correctly specified.

The source is not representable as a floating point number because of an exponent overflow which occurred during data conversion.

Description This is fatal if ED=A and trivial if ED=NA.

User Action Determine whether the field descriptor was correctly specified.

The source is not representable as a floating point number because of an exponent underflow which occurred during data conversion.

Description This is fatal if ED=A and trivial if ED=NA.

User Action Determine whether the field descriptor was correctly specified.

The specified label is not defined.

Description The specified label does not match any label that begins a structured statement in this CREATE_OUTPUT_RECORD statement list.

User Action Either correct the specified label or add the label to the appropriate structured statement.

This directive is not preceded by a SET_INPUT_ATTRIBUTES directive.

Description The SET_INPUT_ATTRIBUTES (or SETIA) directive must precede the other directives.

User Action Place the SET_INPUT_ATTRIBUTES directive before the other directives.

Trailing position is specified in SETSA field descriptor.

Description The sequence field descriptor on the SET_SEQUENCE_ATTRIBUTES directive can specify only the position and length of the sequence field.

User Action Correct the field descriptor.

Trailing position of field precedes starting position of field.

Description This is not permitted.

User Action Correct trailing or starting position.

Unexpected end of directives encountered.

Description The last directive in the file is incomplete.

User Action Append the missing text.

Unexpected {identifier} ignored.

Description FMU ignored the item because it did not belong in this context.

User Action Check the statement; if appropriate, remove the ignored item.

Unknown identifier - {identifier}.

Description FMU does not recognize the value.

User Action Correct the identifier.

Variable {name} is already being used in outer FOR loop.

Description A nested FOR statement cannot specify the same name as a FOR statement in which it is nested.

User Action Change the name on the inner or outer FOR loop.

{item} {item} on FILE {file name} has a length of {number}, exceeding the MAXIMUM_RECORD_LENGTH of {number}.

Description The specified item has a length greater than the MAXIMUM_RECORD_LENGTH (MAXRL) attribute of the file.

User Action Shorten the specified length so that it is within the maximum record length.

{Error} was detected in literal transformation.

Description The format of the literal does not agree with the operand that describes it.

User Action Correct literal or change operand.

Index

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Comments (continued from other side)

Please fold on dotted line;
seal edges with tape only.

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
First-Class Mail Permit No. 8241 Minneapolis, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA
Technology & Publications Division
SVL104
P.O. Box 3492
Sunnyvale, CA 94088-3492

NOS/VE Advanced File Management Usage 60486413 J

We value your comments on this manual. While writing it, we made some assumptions about who would use it and how it would be used. Your comments will help us improve this manual. Please take a few minutes to reply.

Who are you?

- ☐ Manager
- ☐ Systems analyst or programmer
- ☐ Applications programmer
- ☐ Operator
- ☐ Other _____

How do you use this manual?

- ☐ As an overview
- ☐ To learn the product or system
- ☐ For comprehensive reference
- ☐ For quick look-up

What programming languages do you use? _____

_____

How do you like this manual? Check those questions that apply.

Yes	Somewhat	No	
☐	☐	☐	Is the manual easy to read (print size, page layout, and so on)?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Does it tell you what you need to know about the topic?
☐	☐	☐	Is the order of topics logical?
☐	☐	☐	Are there enough examples?
☐	☐	☐	Are the examples helpful? (☐ Too simple? ☐ Too complex?)
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Do the illustrations help you?

Comments? If applicable, note page and paragraph. Use other side if needed.

Would you like a reply? ☐ Yes ☐ No

From:

Name _____ Company _____

Address _____ Date _____

_____ Phone _____

_____

Please send program listing and output if applicable to your comment.