

Advantage™ VISION:Builder®

Advantage™ VISION:Two™

ASL Reference Guide

14.0



Computer Associates™

ALREF140.PDF/D92-002-014

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, the user may print a reasonable number of copies of this documentation for its own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software of the user will have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

© 2002 Computer Associates International, Inc. (CA).

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.



Contents

Chapter 1: Introduction

Operating System and Environment Support	1-1
About This Book.....	1-2
Contacting Computer Associates.....	1-3

Chapter 2: Terminology, Syntax, and Processing

ASL Terminology	2-1
Syntax Terminology	2-2
Spaces.....	2-2
Continuation	2-3
Constants.....	2-3
Names	2-4
Comments	2-6
Arithmetic Expressions	2-6
Logical Expressions.....	2-7
Statement Syntax	2-10
Page Layout.....	2-11
COMBINE Command.....	2-11
The Nature of ASL	2-12
Implicit Loops and Set Operation	2-12
Controlling the Processing of Repeated Segments.....	2-16
Record and Segment Processing	2-17
ASL Code Examples and Usage.....	2-19

Chapter 3: Run Control Command Group

ARRAY Command	3-3
CATALOG Command	3-4
CHECKPOINT Command.....	3-6
COLLATE Command	3-8

CONTROL Command.....	3-10
COPY Command.....	3-18
DEBUG Command.....	3-19
DOCUMENT Command.....	3-21
FILE AUDIT Command.....	3-22
FILE CORDn Command.....	3-23
FILE MASTER Command.....	3-31
FILE REJECT Command.....	3-39
FILE REPn Command.....	3-40
FILE REPORT Command.....	3-41
FILE SUBFn Command.....	3-42
FILE TRAN Command.....	3-45
LINKAGE Command.....	3-47
LISTCNTL Command.....	3-48
LISTLIB GLOSSARY Command.....	3-50
LISTLIB NAMES Command.....	3-51
MULTILIB Command.....	3-52
OVERRIDE Command.....	3-53
OWNCODE Command.....	3-54
RETRIEVE Command.....	3-55
ROUTE Command.....	3-57
TRACK Command.....	3-59
WORK Command.....	3-61

Chapter 4: Procedural Command Group

Built-In Functions.....	4-3
Value Functions and Conditional Functions.....	4-3
Specifying Functions.....	4-3
Types of Built-In Functions.....	4-3
FIND Function (Conditional).....	4-5
LOCATE Function (Conditional).....	4-7
LOOKUP Function (Value).....	4-8
PF Function (Value).....	4-11
SCAN Function (Conditional).....	4-13
VALIDATE Function (Conditional).....	4-16
CALL Command.....	4-18
CASE Command.....	4-22
COMBINE Command.....	4-23
CONTINUE Command.....	4-24
DO Command.....	4-25

ELSE Command	4-30
END Command	4-32
FIELD Command	4-33
GO TO Command	4-36
IF Command	4-38
INCLUDE Command	4-40
LEAVE Command	4-42
LET Command	4-44
LOCATE Command.....	4-48
PROC Command.....	4-50
RELEASE Command	4-55
REPLACE Command	4-57
RETURN Command	4-59
TRANSFER Command.....	4-60

Chapter 5: Report Command Group

AVERAGE Command.....	5-3
COMPUTE Command.....	5-4
COUNT Command	5-7
CUMULATE Command.....	5-8
DATA Command.....	5-9
END Command	5-12
FORMAT Command.....	5-13
GROUP Command	5-26
ITEM Command.....	5-28
LINE Command	5-31
MAX Command	5-34
MIN Command	5-35
NEWPAGE Command	5-36
ORDER Command.....	5-37
PERCENT Command	5-38
PREFACE Command	5-40
RATIO Command	5-41
REPORT Command	5-42
SECTION Command	5-50
SIZE Command	5-51
SKIP Command	5-52
TITLE Command.....	5-53
TOTAL Command.....	5-54

XREP Command.....	5-55
Diagnostic Messages.....	5-65

Chapter 6: Subfile Output Command Group

EXTRACT FILE Command.....	6-2
EXTRACT DDNAME Command	6-9
EXTRACT DBDNAME Command.....	6-14
EXTRACT TABLE Command.....	6-18

Chapter 7: ASL Examples

Appendix A: ASL Quick Reference

Terminology	A-1
References.....	A-2
Constants.....	A-2
Names	A-2
Qualifiers	A-3
Comments	A-3
Arithmetic Expressions	A-3
Logical Expressions.....	A-4
Statement Syntax.....	A-4
Continuation	A-5
Built-In Functions.....	A-5
Conditional Functions	A-5
Value Functions	A-5
Commands	A-6

Appendix B: Relationship of ASL Statements to Fixed-Format-Syntax Statements

Appendix C: Technical Notes

Reserved Words.....	C-1
Qualifiers	C-4

Patterns	C-4
Validation Patterns.....	C-4
Edit Patterns.....	C-5
Character String Data	C-6
Numeric Data (Packed, Zoned, and Fixed Point Binary Only).....	C-6
Rules for Edit Patterns.....	C-7
Output Edit	C-9
Commas.....	C-9
Standard Notation.....	C-9
Floating/Edit Suppress	C-9
Float (Floating-Edit-Char).....	C-10
Fill (Fill-Edit-Char).....	C-10
Trail (Trailing-Edit-Char).....	C-11
Edlen (Edit-Length)	C-11
Valid Field Types and Default Field Lengths	C-12

Appendix D: Flags

ASTATUS Flag	D-7
CHKP Flag (IMS Only).....	D-9
CKPTID Flag (IMS Only)	D-9
COLUMN Flag	D-9
COMMAND Flag (GDBI Only).....	D-10
CONDCODE Flag	D-11
CSTATUS Flag	D-12
DATE Flag	D-12
DELETE Flag (VISION:Builder 4000 Model Series Only)	D-13
ECORD Flag	D-13
EOF Flag	D-15
FDNAME Flag (GDBI Only).....	D-16
FILE Flag (GDBI Only)	D-16
FILEID Flag (GDBI Only).....	D-16
ISDATE Flag	D-16
JULANX Flag	D-17
JULIAN Flag	D-17
LILIAN Flag	D-17
LNUMBER FLAG	D-18
LSTART Flag	D-18
LSTATUS Flag	D-19
M4AUDIT Flag (VISION:Builder 4000 Model Series Only).....	D-21
M4CORDn (n=1 to 9) Flags	D-21

M4NEW Flag (VISION:Builder 4000 Model Series Only)	D-21
M4OLD Flag	D-21
M4REJECT Flag (VISION:Builder 4000 Model Series Only)	D-22
M4SUBFn (n= 0 to 9) Flags	D-22
M4TRAN Flag (VISION:Builder 4000 Model Series Only)	D-22
MISSPASS Flag	D-22
MNUMBER Flag.....	D-23
MODE Flag (GDBI Only)	D-23
MSTART Flag.....	D-24
MSTATUS Flag (GDBI Only)	D-25
OWN Flag.....	D-25
PAGE Flag	D-26
PASSWORD Flag (GDBI Only)	D-26
RESTART Flag (IMS Only).....	D-26
RETURNCD Flag	D-26
RNUMBER Flag	D-27
ROW Flag	D-27
RSTART Flag	D-28
RSTATUS Flag	D-29
SEGNAME Flag (GDBI Only)	D-30
SQL Flag (DB2 Only)	D-30
SSCOUNT Flag	D-30
STRAN Flag (VISION:Builder 4000 Model Series Only)	D-31
TIME Flag	D-32
TODAY Flag	D-32
TODAYX Flag	D-32
TRAN Flag (VISION:Builder 4000 Model Series Only)	D-33
XTRAN Flag (VISION:Builder 4000 Model Series Only)	D-33

Appendix E: Conversion Functions

Why ASL?.....	E-1
Defining ASL Procedures.....	E-1
Tips and Techniques.....	E-2
ASL Syntax and Terminology.....	E-3
Conversion Examples	E-4
Conversion Table.....	E-14

Index

ASL (Advanced Syntax Language) is a free-form language used to build applications consisting of run control, procedural, reporting, and subfile output statements. It is designed with both the programmer and the end user in mind. The syntax and structure of ASL statements are similar in nature to other free-form languages such as C and COBOL. However, ASL goes beyond these languages by providing many simple yet powerful business-oriented functions.

The power of ASL is illustrated in how simple it is to change text in a field. For example, `REPLACE STRING 'ABC' IN NAME WITH 'XYZ'` does just what it says. Each time ABC is found in the field NAME, ABC is replaced with XYZ.

Field NAME contents:

Before: ' THE ABC COMPANY'

After: ' THE XYZ COMPANY'

Because the statements that are used to build ASL procedures are structured like sentences, the language is easy to understand, self-documenting (although comments are optional), and easy-to-maintain.

Operating System and Environment Support

ASL is directly supported in its native free-form syntax when using VISION:Workbench™ for DOS and VISION:Workbench for ISPF (also known as the Definition Processor).

In OS/390 and z/OS

When using the VISION:Builder®, VISION:Inform®, and VISION:Two™ engines, the functionality of ASL is directly supported in its native free-form syntax in the OS/390® and z/OS™ environments.

In CMS and VSE

Only the Procedural, Reporting and Subfile Command group statements of ASL are supported by the engines in the CMS and VSE environments.

For VISION:Builder and VISION:Inform users in the CMS and VSE environments, VISION:Workbench for DOS translates your native ASL syntax coding into fixed format coding during the export process. See the VISION:Workbench for DOS Setup window description in the [VISION:Workbench for DOS Reference Guide](#) for details on specifying target host support of native ASL.

About This Book

Note: Text that is specific to a host engine is indicated by a note.

This book contains information specific to the use and operation of ASL and assumes that you are familiar with one of the host engines, VISION:Builder, VISION:Inform, or VISION:Two.

[Chapter 1, Introduction](#)

Describes this book.

[Chapter 2, Terminology, Syntax, and Processing](#)

Starts with information about how to read the syntax of the statements and defines some common computer terminology with respect to ASL.

[Chapter 3, Run Control Command Group](#)

Describes the commands that specify basic run parameters and controls, file/database parameters and controls, and catalog related operations.

[Chapter 4, Procedural Command Group](#)

Describes the procedure statements. Again, the syntax is displayed and defined. Examples are given to illustrate the procedure statement.

[Chapter 5, Report Command Group](#)

Describes the commands that are used to specify report output.

[Chapter 6, Subfile Output Command Group](#)

Describes how to use the EXTRACT commands.

[Chapter 7, ASL Examples](#)

Shows examples using ASL.

[Appendix A, ASL Quick Reference](#)

Contains all of the ASL commands.

[Appendix B, Relationship of ASL Statements to Fixed-Format-Syntax Statements](#)

Relates the functions and procedure statements to their host engine counterparts.

[Appendix C, Technical Notes](#)

Contains technical information defining some of the operand entries. When necessary, this appendix will be referenced in the explanation of the operand. It also contains a reserved word list.

[Appendix D, Flags](#)

Lists all of the host engine flags.

[Appendix E, Conversion Functions](#)

Provides suggestions for converting existing VISION:Builder requests to ASL procedures.

Contacting Computer Associates

For technical assistance with this product, contact Computer Associates Technical Support on the Internet at esupport.ca.com. Technical support is available 24 hours a day, 7 days a week.

Terminology, Syntax, and Processing

This chapter defines the terms that are used to describe ASL statements, ASL syntax, and how ASL processes.

ASL Terminology

ASL is a free-form language consisting of commands and functions. A statement begins on a new line and consists of an optional label followed by a command. If the statement has a label, the label must be followed immediately (without intervening spaces) by a colon. What appears on the rest of the statement after the command depends on the syntax of the command.

For :

```
LABELX: LET FIELDA = FIELDB
```

is an actual procedure statement where:

LABELX	Specifies the statement label.
LET	Specify the command.
FIELDA = FIELDB	Places the contents of field B into field A.

Syntax Terminology

The following terminology is used to describe the elements of an ASL statement.

TERM	DESCRIPTION
Label	A label identifies a specific statement. The label is optional. If used, place the label before a procedure statement and follow the label immediately (without intervening spaces) by a colon (:).
Command	A command identifies the kind of procedure statement. You can follow a command by keywords, functions, constants, names, expressions, and comments.
Function	A function is an operation or declaration that derives a value or condition from other data. Functions may contain elements such as keywords, names, and constants.
Keyword	A keyword identifies how the following statement elements are interpreted. Many keywords are optional.
Keyword operand	A keyword operand (or operands) identifies the data values associated with a keyword. In the statement, follow a keyword with one or more spaces and a keyword operand. For example: keyword1 operand keyword2
Keyword phrase	A keyword phrase is the name given to the combination of a keyword plus its operands.

Spaces

Use one or more spaces to separate the command, keywords, and operands, unless otherwise stated (syntax rules) or unless a special character (such as an arithmetic operator) is present.

The following is an example of an ASL command using operators as separators between the operands on the IF command.

```
IF A+B=C+D THEN
```

To make the procedure statement easier to read, use blanks between the operands and operators. For example:

```
IF A + B = C + D THEN
```

Continuation

You can write statements on multiple lines. Terminate each line (ignoring comments) by an optional blank space followed by a comma. Continue the remainder of the statement on the following lines. This is useful for clarity. For example:

```
IF      NAME    =      'THE ABC COMPANY' ,
AND    NUMBER  =      '00001' ,
OR     NUMBER  =      '00002' ,
OR     NUMBER  =      '00003' ,
THEN
```

Commas may also be used within a statement to improve readability. For example:

```
FILE MASTER INPUT, NAME EMPLOYEE, ACCESS SEQUENTIAL
```

Constants

There are six types of constants: character, integer, decimal, floating point, time, and pattern. Each type of constant is described in the following pages.

Character Constants

Delimit character constants with single quotation marks. For example:

The Value	The Character Constant Representation
A	'A'
	' '
123	'123'
THE X	'THE X'
CAN'T	'CAN' 'T'

Specify a single quotation mark within a constant with two consecutive single quotation marks.

Integer Constants

Specify integer constants with only numeric digits only.

- If the integer is negative, place a minus (-) sign before the integer constant.
- If the integer is positive, place an optional plus sign (+) before the integer constant. For example:

```
+100
-2
0
```

Decimal Constants

Specify decimal constants with numeric digits, an optional sign, and a decimal point.

If you use a plus (+) sign or a negative (-) sign, make it the first character in the constant. For example:

```
-200.0  
 3.99  
-.05  
+123.456
```

Floating Point Constants

Specify floating point constants with an optional sign preceding an integer or decimal constant followed by a power of ten expressed in exponential notation.

For example:

```
1.50E10  
13.75E-5  
-5200E+5
```

Time Constants

Specify a time constant, HH:MM:SS.n...n (hours, minutes, seconds, decimal seconds), by the letter T, followed by a beginning single quotation mark, the time constant, and a single closing quotation mark. For example:

```
T' 23:16:11'  
T' 12:01:00.125'  
T' :09:10.5'
```

Patterns

Specify a pattern starting with the letter P, followed by a beginning single quotation mark, the string of special symbols, and ending with the closing single quotation mark. For example:

```
P' ZZZ999'  
P' # (#999#) #999#-#9999'
```

For more information on pattern symbols, see [Appendix C, Technical Notes](#).

Names

- Begin field names and statement labels with an alphabetic character (A-Z).
- Make the remaining characters either alphabetic characters (A-Z), numeric digits (0-9), national characters (\$, #, @), or underscores (_).
- Enclose field names that do not conform to this syntax in double quotation marks.
- Enclose names that conflict with keywords of the statement in double quotation marks.
- Field names may be up to 30 characters long; statement labels may be up to 8 characters long.

The following are valid field names or procedure statement labels:

```
CUST_NUM
BAL30
CUSTNO
"AMT-DUE"
```

Qualifiers

You can prefix a name with a standard 1-character qualifier and a period. A qualifier identifies a specific file or usage of a field. Qualifiers relate fields to their origins:

Qualifier	Type of Location	
Blank or N	New master file.	
1-9	Coordinated files 1-9.	
T	Temporary file.	
F	Flag field.	
X	Transaction file	} For VISION:Builder and VISION:Two.
O or 0	Old master file.	
W	Working storage	
V	Linkage section.	
A, B, E, G, H, J, K, M, Q, 1-9	Array (must match the qualifier that identifies the array.	

The following are valid qualified field names:

```
T . TEMP
N . CUST_NUM
1 . CORD_FLD
F . DATE
```

Example 1

```
T . TEMP
```

This is a reference to a temporary field named TEMP.

Example 2

```
1 . CORDFLD
```

This is a reference to a field named CORDFLD from the file in the application that has been assigned the qualifier 1.

[Appendix C, Technical Notes](#) contains a list of all valid qualifiers and their origins.

Names with Special Characters

Enclose names that include special characters in double quotation marks. The following is an example of a field name with a special character (an embedded blank):

```
N."DATE ONE"
```

Without the special notation, the blank in the field name is interpreted as two separate fields instead of one.

Comments

Place comments anywhere following a semicolon (;), except on a continued line. For example:

```
;      RESETTING A TEMPORARY FIELD
;
LET T.REGION = '      ' ; RESET FIELD TO BLANKS
```

The first two lines show comments on lines by themselves. The third line is an example of a comment on a command line. Separate the comment from the command and its keyword phrase by a semicolon.

Arithmetic Expressions

Code arithmetic expressions using the operators +, -, *, and / for addition, subtraction, multiplication, and division, respectively. For example:

```
A+B   A-B   B*C   D/C
```

As in conventional algebraic notation, operations within an arithmetic expression are executed from left to right. However, multiplication and division are performed prior to addition and subtraction unless this order is overridden by parentheses. For example:

```
A-B*C+D
```

In this arithmetic expression, the multiplication between field B and field C is performed before the addition and the subtraction.

When you write the expression as $(A-B) * (C+D)$, the addition and subtraction are performed before the multiplication.

To evaluate the difference, assume that:

```
A=5
B=3
C=2
D=1
```

for the two arithmetic expressions given in the prior example. [Figure 2-1](#) shows the steps taken to come up with the result of the arithmetic expressions.

<u>Without Parentheses</u>	<u>With Parentheses</u>
A-B*C+D	(A-B) * (C+D)
5-3*2+1	(5-3) * (2+1)
5-6+1	2 * 3
0	6

Figure 2-1 Steps Taken to Achieve the Result of the Arithmetic Expressions

Logical Expressions

Use logical expressions in IF, CASE, and DO commands.

Make a logical expression from conditional functions, relational expressions, or list expressions connected by logical operators.

Conditional Functions

A conditional function is a function that returns a true or false condition. For example:

```
VALIDATE (FIELD ORDRDATE DATE)
```

shows the VALIDATE function. The date validation of the field ORDRDATE will either be true or false depending on the contents of ORDRDATE being a valid date.

In the following examples, A can be a field name, and B and C can be any constant, field name, or arithmetic expression.

Relational Expressions

A relational expression is composed of two values connected by a logical operator. In relational expressions, you can represent logical operators as characters or as symbols:

```
EQ or =
NE or <>
GT or >
LT or <
GE or >=
LE or <=
```

Examples

```
A EQ B (equal)
A NE B (not equal)
A LT B (less than)
```

A GT B (greater than)
A LE B (less than or equal to)
A GE B (greater than or equal to)

List Expression

A list expression lists the specific values to test. For example, `NUMBER EQ 00001 00002 00003 00004` specifies that `NUMBER` should be compared against the four values listed.

In a list expression, use only the `EQ` and `NE` operators. In relational or list expressions, you can represent logical operators as characters or as symbols:

`EQ` or `=`
`NE` or `<>`
`GT` or `>`
`LT` or `<`
`GE` or `>=`
`LE` or `<=`

Boolean Logical Operators

There are three Boolean logical operators: `AND`, `OR`, and `NOT`. Use parentheses to specify the order of evaluation.

Example 1 The OR operator.

`A=B OR A=C`

If either relational expression is true, the logical expression is true.

Example 2 The AND operator.

`A=B AND A=C`

Both relational expressions must be true for the logical expression to be true.

Example 3 The NOT operator.

`NOT (A=B AND A=C)`

Both relational expressions must be true for the logical expression to fail. The whole expression is true if the expression in parentheses is false. In other words, the `NOT` operator reverses the evaluation of true or false for the final outcome of the expression.

The `NOT` operator is best understood when used in a conditional function such as `VALIDATE`. For example, `NOT VALIDATE(DUEDATE DATE)` is true when the field `DUEDATE` is not valid.

You can also use the NOT operator to exclude a few values that are not wanted for processing instead of including all of the values that are wanted. For example, the following is true for all values of NUMBER except 1 or 5:

NOT (NUMBER = 1 OR NUMBER = 5)

The following table shows numeric examples and the subsequent true or false evaluations of logical expressions.

	A=B	A=C	A=B OR A=C	
A=1, B=2, C=2	FALSE	FALSE	FALSE	
A=2, B=2, C=1	TRUE	FALSE	TRUE	
A=2, B=2, C=2	TRUE	TRUE	TRUE	

	A=B	A=C	A=B AND A=C	NOT (A=B AND A=C)
A=1, B=2, C=2	FALSE	FALSE	FALSE	TRUE
A=2, B=2, C=1	TRUE	FALSE	FALSE	TRUE
A=2, B=2, C=2	TRUE	TRUE	TRUE	FALSE

Statement Syntax

This section describes the conventions used to provide a precise description of the syntax of a function or command.

Enter commands and functions in the exact order given on the syntax line.

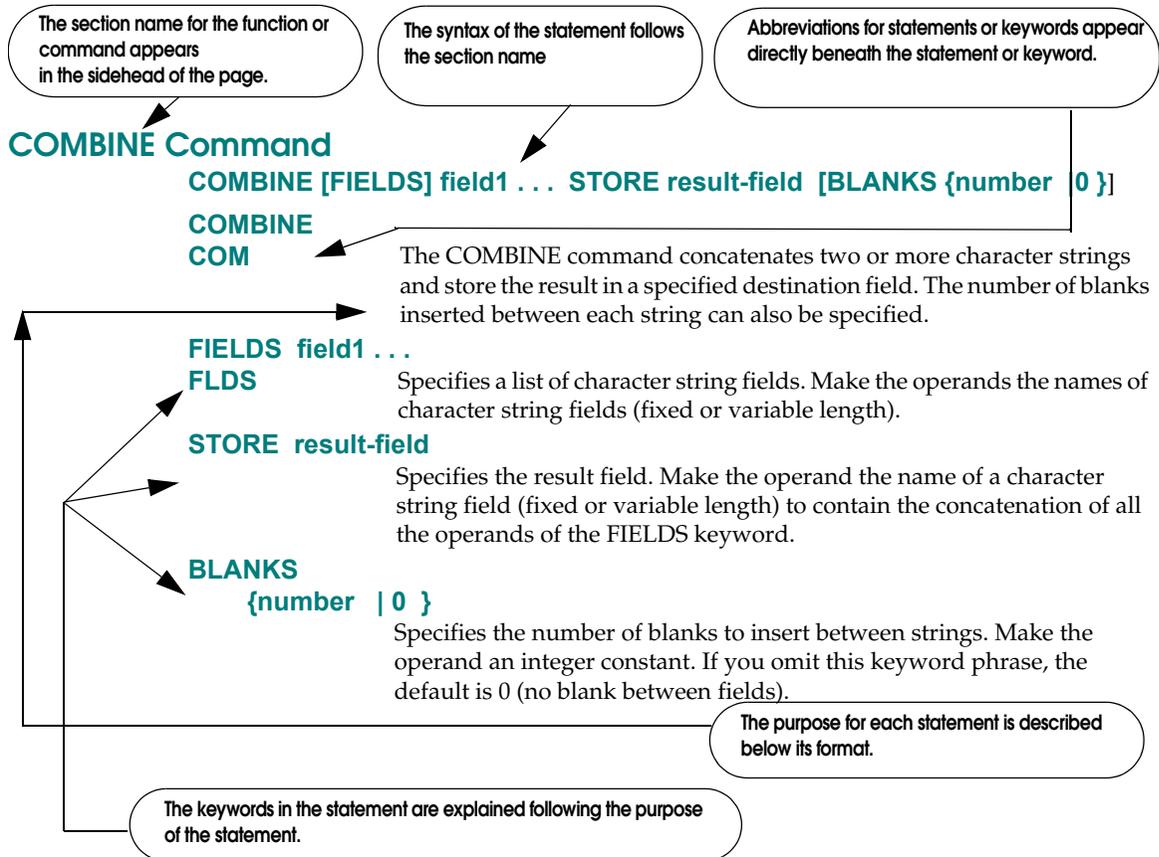
The following syntax conventions apply to ASL statement syntax. As the examples given with each command show, you enter your procedures using uppercase letters.

- Brackets [] indicate an optional keyword or parameter.
- Braces { } indicate a choice of entry; unless a default parameter is indicated by an underscored entry, you must choose one of the entries.
- Required parameters do not have brackets or braces surrounding them.
- Items separated by a vertical bar (|) represent alternative items. Select only one of the items.
- Items separated by a plus sign (+) represent multiple items. Select as many as needed.
- An ellipsis (. . .) indicates that you can use a progressive sequence of the type immediately preceding the ellipsis. For example: name1, name2, ...
- Bold, uppercase type indicates the characters to be entered. Enter such items exactly as illustrated. You can use an abbreviation if it is indicated.
- Italic, lowercase type specifies fields to be supplied by the user.
- Underscored type indicates a default option. If you omit the parameter, the underscored value is assumed.
- Separate commands, keywords, and keyword phrases by blanks, commas, or both.
- Enter punctuation such as parentheses and colons exactly as shown.

Note: The section heading for each function and command appear at the top of a page.

Page Layout

The following describes the format used to present each function or command.



Example

```
COMBINE FIRSTNAM LASTNAME STORE NAME BLANKS 1
```

The Nature of ASL

ASL enables you, the application developer, to work with a logically related set of data. The objective is for the application developer to focus upon the algorithms and business considerations of the problem independently of the way the data is physically structured or organized. Thus, for the most part, the application developer can write ASL statements without concern for the structure of a record and without concern as to how the data is accessed.

Underlying ASL, therefore, are several important considerations that enable the application developer to process the data correctly and consistently.

In developing an application in ASL, the application developer merely references fields by name. The ASL processor locates all fields that are applicable. More importantly, the ASL processor ensures that when data is requested from several different fields, and even from different databases, only the correctly related data is made available. In this manner, only correct and consistent sets of data are processed.

The application developer should be aware of the following important and powerful capabilities built into the ASL processor and the ASL language:

- Implicit loops or set operation.
- Record and segment processing.

Implicit Loops and Set Operation

ASL is a set language. This means that one statement of ASL processes all occurrences of the data (that is, the complete set of occurrences). In a structured record, repeated segments of data represent different occurrences of the data. In most business situations, the application developer wants to apply a business process (or algorithm) to all occurrences of the data, or at least to specifically selected occurrences.

The ASL processor actually applies the ASL statements to each set of data in turn. The application developer can write the algorithm as if it applied to only one instance of the data. In practice, all instances are selected.

Consider the simple structure shown in [Figure 2-2](#).

Field Name	DEPT	MANAGER	MGRSAL	Level 1
	SALES	GREEN	60,000	

Field Name	NAME	SALARY	SALGRADE	Level 2
	SMITH	40,000	5	
	JONES	50,000	6	
	WHITE	40,000	5	
	BROWN	65,000	7	

Figure 2-2 Employee File

This structure is typical of a department level with subordinate segments, one for each employee. To increase the salary of everybody in the department by 5 percent, use the following ASL statement:

```
LET SALARY = 1.05*SALARY
```

While this looks like one statement, it carries with it the implied “for all occurrences of SALARY.”

The result of this one statement is shown in [Figure 2-3](#).

Field Name	DEPT	MANAGER	MGRSAL	Level 1
	SALES	GREEN	60,000	

Field Name	NAME	SALARY	SALGRADE	Level 2
	SMITH	42,000	5	
	JONES	52,500	6	
	WHITE	42,000	5	
	BROWN	68,250	7	

Figure 2-3 Update All Records in the Employee File

The single statement acted on all occurrences of the data.

If you have your original data of [Figure 2-2](#), but you need to increase the salaries of those people whose salary grade is less than 7, use the ASL statements shown below.

```
IF SALGRADE LT 7
  LET SALARY = 1.05*SALARY
END
```

The result of this statement is shown in [Figure 2-4](#).

Field Name	DEPT	MANAGER	MGRSAL	Level 1
	SALES	GREEN	60,000	

Field Name	NAME	SALARY	SALGRADE	} Level 2
	SMITH	42,000	5	
	JONES	52,500	6	
	WHITE	42,000	5	
	BROWN	65,000	7	

Figure 2-4 Update Selected Records in the Employee File

The ASL processor identifies all segments of data that are relevant and executes the statement. As the application developer, you do not have to write any complex looping structures.

As another example, assume that the three level record structure shown in [Figure 2-5](#) exists.

Field Name	DIVISION			
	WEST			Level 1

Field Name	DEPT	MANAGER	MGRSAL	
	SALES	GREEN	60,000	Level 2

Field Name	NAME	SALARY	SALGRADE	} Level 3
	SMITH	40,000	5	
	JONES	50,000	6	
	WHITE	40,000	5	
	BROWN	65,000	7	

Field Name	DEPT	MANAGER	MGRSAL	
	FINANCE	BLACK	45,000	Level 2

Field Name	NAME	SALARY	SALGRADE	} Level 3
	THOMAS	50,000	6	
	MILLS	33,000	4	
	NEWS	40,000	5	

Figure 2-5 Three Level Record Structure

To select every employee who earns more than their manager, use the following ASL statements:

```
IF SALARY GT MGRSAL
  CALL REPORT HIGHPAID
END
```

These statements select the following sets of data for the report HIGHPAID:

BROWN	65,000	7
THOMAS	50,000	6

In processing these ASL statements, the ASL processor maintains the integrity of the relationships of the data. Each employee is associated with the appropriate manager. No employee is associated with somebody else's manager. The data that is returned and processed is that which you logically would expect.

ASL, therefore, is a set language that processes all occurrences of the data that satisfy any existing criteria. ASL does not stop with the first occurrence of the data. Any procedural statements are processed against each occurrence of the data that satisfies the selection criteria. When this is done, ASL searches for the next occurrence of the data that satisfies the selection criteria. This process occurs until all instances of the data have been processed. Some application developers may like to think of this as the creation of an implicit loop based upon the number of segment occurrences that exist.

In the vast majority of cases, application developers can write an algorithm or business consideration as if there were only one occurrence of the data and be secure in the knowledge that all appropriate occurrences will be processed.

The examples show that one or two statements can process large amounts of data. There are no complicated data navigation commands; it was all transparent. You can focus on the problem and leave the data to ASL.

In some instances, you must know that you are, in fact, dealing with a set of data and not an individual occurrence. Under some circumstances, the following specific situations can occur:

- Stop the procedural algorithm working on the particular occurrence of the data and move on to the next occurrence.
- Stop all processing of any future occurrences of the data.

Sometimes these situations are relevant to the particular problem being solved, while in other cases there are important performance considerations.

When processing large structured records, for example, you can improve performance by bypassing processing of segments where it is known that no action is required.

Controlling the Processing of Repeated Segments

ASL provides commands for the application developer to control the processing of repeated segments within a structure. Two particular commands are of importance: CONTINUE and LEAVE.

- The CONTINUE command enables the application developer to stop the algorithm working on the current occurrence of the data and move to the next occurrence.
- The LEAVE command causes the procedure to stop processing this occurrence of the data and to bypass all other occurrences of the data within the current set for the current procedure.

Within ASL, any selection statement (such as IF, DO) limits the available segments of data for processing by subsequent procedural statements. This applies also to subroutine procedures, reports, or subfiles. Any object that is called from another object inherits the same view of the data as currently defined by the parent object. All segments beyond the scope of this view are still available for processing. Once again, the consistency of data is maintained.

Record and Segment Processing

The set theory philosophy, as described above, says that the application developer should not worry about how many occurrences of the data exist nor the relationships between the various pieces of data. That is the responsibility of ASL and the ASL processor. The ASL processor guarantees that all occurrences of the data are available for processing with the correct relationships between the data.

This philosophy continues in ASL with the philosophy of handling records and segments within a record. The ASL user does not have to be concerned with where the data physically exists. ASL is structured to be able to handle records and segments in a similar manner.

Records are normally thought of as separate physical entities; whereas, segments are thought of as being contained within a record and being repetitions of data.

The Distinction Between Records and Segments

The distinction between records and segments is an artificial one imposed upon the application developer by virtue of hardware constraints. Many records in a file really constitute repeats of root segments. In effect, they are no different from repeats of subordinate segments. Unfortunately, because of the difference in nature of the physical arrangement of the data, it frequently calls for different programming techniques to access the data. ASL and the ASL processor remove these restraints from the application developer.

For the most part, you, the application developer, can write your application as if you were dealing with an infinite space of data. All data is treated as if the fields belong to a segment. It is the responsibility of ASL and the ASL processor to work out whether accessing a segment is merely an act of navigating through a structured record or whether a physical read of a new record from the file must be made. All of this is transparent to the application developer.

Automatic Navigation

Navigation through records and segments is automatic for a majority of applications. As described above, a reference to a field causes navigation through a record for all suitable occurrences of segments containing that data. When all segments have been processed, the ASL processor automatically reads the next record and transfers control to the beginning of the processing cycle.

Application-Controlled Navigation

Under certain circumstances, an application developer might not want to have the ASL processor perform the automatic navigation through the data. If this is the case, the application developer can use explicit functions (for example, FIND) to locate specific instances of segments.

When you issue a FIND command for a segment, the ASL processor locks onto the segment that is identified or located. All subsequent procedural statements can only access data within that segment. If you require access to all the occurrences of the segment after a FIND command completes, you must issue a RELEASE command.

ASL provides powerful commands for those applications where the application developer needs to control the navigation through the data.

ASL Code Examples and Usage

The following is an example of a simple VISION:Builder application written entirely in ASL.

```

; ASL Run Control Group for Sample Application
; Application uses a Master File and one Coordinated File
;
CONTROL TERM, DB2 D61A INM4CALL, RPTMSG NO
FILE REPORT
;
; Declare Master file with File Definition from SQL clause
FILE MASTER INPUT, KEYS NONE,
  SQL "SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, ",
      "BIRTHDATE, HIREDATE, SALARY FROM DSN8610.EMP "
;
; Declare Coordinated file with File Definition from COMLIB
FILE CORD1 NAME TDEPT, DIRECT BY O.WORKDEPT
;
; End of ASL Run Control
;
;
; Begin ASL Procedure Group
; First procedure contains processing logic and 2 reports
; A third report outputs data in CSV format
;
PROC INFO 'Main Procedure'
NAME: FIELD V 22 ;Temporary field to hold combined name
;
; Combine Elements of Name into One Field
COMBINE LASTNAME ', ', BLANKS 0, STORE T.NAME
COMBINE T.NAME FIRSTNME MIDINIT, BLANKS 1, STORE T.NAME
;
REPORT EMPNO, T.NAME, BIRTHDATE
  TITLE 'Report Showing Birth Dates of all Employees'
  ORDER BY EMPNO
  ;Use LE Picture to display Birth Date with Day of Week
  ITEM BIRTHDATE PIC P'Wwwwwwwwz, Mmm DD, YYYY'
  FORMAT DATEFMT DATE, WIDTH 80
END REPORT
;
REPORT 1.DEPTNO, EMPNO, T.NAME, HIREDATE, SALARY
  TITLE 'Report Showing Employees by Department'
  ORDER BY 1.DEPTNO EMPNO
  GROUP BY 1.DEPTNO 1.DEPTNAME, SUBTITLE
  ITEM EMPNO SPACES 7
  FORMAT WIDTH 80
END REPORT
;
END PROC
;
REPORT BIRTHDATE HIREDATE SALARY
  FORMAT METHOD CSV, DDNAME CSVOUT1
END REPORT
;
; End of Application

```

The use of ASL results in easy-to-read, English-like, application code.

Combining ASL with Traditional Syntax Coding

Existing applications using fixed-syntax statements may be converted incrementally by coding in ASL only those sections of the application that are being revised or enhanced. To assist in this process, the ASL processor recognizes two special directives that may be used to combine both ASL and fixed-syntax statements in the same application. These directives are:

```
;;BEGASL
```

and

```
;;ENDASL
```

The `;;BEGASL` directive may be used to indicate that the following statements are ASL rather than fixed-syntax. The `;;ENDASL` directive may be used to indicate that the following statements are fixed-syntax rather than ASL. Both of these directives must begin in column 1 of a statement. These directives should only be used at the boundaries of logical coding units such as:

- after ASL run control and prior to fixed-syntax run control or fixed-syntax requests
- after fixed-syntax run control or requests and prior to an ASL procedure, report, or subfile

The following example illustrates the use of the `;;BEGASL` and `;;ENDASL` directives.

```
DB2RUN2 RCTEMPL  S U  S  #T  R
DB2RUN2 RPDB2   D61A  INM4CALL
DB2RUN2 RPLISTCNTL  N
;;BEGASL
PROC
LILDATE:  FIELD D  4
FEEDBACK: FIELD C 12
WEEKDAY:  FIELD C  80
NAME:     FIELD V  22
;
; Combine Elements of Name into One Field
COMBINE LASTNAME, ',' STORE T.NAME
COMBINE T.NAME, FIRSTNAME, MIDINIT BLANKS 1 STORE T.NAME
;
; Convert Birth Date to a Lilian Date using Picture 'YYYY-MM-DD'
CALL CEEDAYS USING BRTHDATE, 'YYYY-MM-DD',
                T.LILDATE, T.FEEDBACK
;
; Convert Lilian Date to a Format Showing Day of the Week
CALL CEEDATE USING T.LILDATE, 'Wwwwwwwwwz, Mmm DD, YYYY',
                T.WEEKDAY, T.FEEDBACK
;
CALL REPORT REPORT1          ;Output Report
END PROC
;;ENDASL
REPORT1 ERTODAY              S
REPORT1 E1                   F
REPORT1 R1                   EMPNO
REPORT1 R1                   TNAME
REPORT1 R1                   BRTHDATE
REPORT1 R1                   TLILDATE
REPORT1 R1                   TWEEKDAY
REPORT1 T1                   Report Showing Use of LE Routines CEEDAYS and CEEDATE#
                                0124
```

Alternatively, the ASL COPY FIXED command can be used to copy in fixed-syntax statements as appropriate. The COPY FIXED command is particularly useful where a fixed-syntax file definition, prepared by the Workbench for ISPF and stored in a definition library (PDS), needs to be included as an in-stream file definition. The following example illustrates the use of the COPY FIXED command.

```
CONTROL TERM, DB2 D61A INM4CALL, RPTMSGS NO
;
; Declare Master, Coordinated, and Report files
FILE MASTER INPUT, NAME TEMPL, KEYS NONE
FILE CORD1 NAME TDEPT, DIRECT BY O.WORKDEPT
FILE REPORT
;
COPY COPYLIB(TEMPL) FIXED
COPY COPYLIB(TDEPT) FIXED
;
MAIN: PROC
LILDATE: FIELD D 4
FEEDBACK: FIELD C 12
WEEKDAY: FIELD C 80
NAME: FIELD V 22
;
; Combine Elements of Name into One Field
COMBINE LASTNAME, ', ' STORE T.NAME
COMBINE T.NAME, FIRSTNAME, MIDINIT BLANKS 1 STORE T.NAME
;
; Convert Birth Date to a Lilian Date using Picture 'YYYY-MM-DD'
CALL CEEDAYS USING BRTHDATE, 'YYYY-MM-DD',
T.LILDATE, T.FEEDBACK
;
; Convert Lilian Date to a Format Showing Day of the Week
CALL CEEDATE USING T.LILDATE, 'Wwwwwwwwz, Mmm DD, YYYY',
T.WEEKDAY, T.FEEDBACK
;
CALL REPORT REPORT1 ;Output Report
END PROC
;
; Copy Fixed-Syntax Request
COPY COPYLIB(REPORT1) FIXED
;
; End of Application
```

Use of ASL Prior to VISION:Builder 14.0

Prior to this release of VISION:Builder, ASL could only be used for procedures as in the example below.

1. Place ASL statements after an ER statement.
2. On the next line, place two system delimiters followed by PROC, starting in column one.
3. Enter the ASL procedure.
4. Place two system delimiters followed by PEND, starting in column one on the line after all of the ASL statements for the procedure.

[Figure 2-6](#) is an example of in-stream ASL procedural statements as supported prior to this release.

```
procname ER
##PROC
STOCKVAL: FIELD P 5 DEC 2
IF QANONHND > 0
  LET T.STOCKVAL = QANONHND * UNITCOST
  CALL REPORT VALUEREP
END
##PEND
```

Figure 2-6 In-Stream ASL Statements

You can use ASL procedures and traditional requests (PR statements) in the same application. However, you cannot use PR statements and ASL in the same procedure or request.

Applications coded in this manner will continue to work.

Capturing the Equivalent Fixed-Syntax Statements

The equivalent fixed-syntax statements for any ASL code may be captured by including a DD statement with a DDNAME of M4FIXED in the job-step JCL. This output file will contain fixed-length 80-byte records suitable for use as M4INPUT statements if so desired.

See the *VISION:Builder Environment Guide* for more information regarding the use of the M4FIXED DD statement.

Run Control Command Group

The Run Control command group contains the commands that specify basic run parameters and controls, file/database parameters and controls, and catalog related operations.

The following table identifies each of the commands in the run control group and their function:

Command Name	Function
ARRAY	Internal array parameters
CATALOG	Request cataloging controls
CHECKPOINT	Checkpoint parameters and controls
COLLATE	Report collating parameters and controls
CONTROL	Application parameters and controls
COPY	Copy commands not in the primary input stream
DEBUG	Debugging output controls
DOCUMENT	Application documentation controls
FILE AUDIT	Audit file parameters
FILE CORDn	Coordinated file and external array parameters and controls
FILE MASTER	Master file parameters and controls
FILE REJECT	Reject file parameters
FILE REPn	Additional report file parameters
FILE REPORT	Primary report file parameters
FILE SUBFn	Subfile file parameters
FILE TRAN	Transaction file parameters and controls

Command Name	Function
LINKAGE	Linkage section parameters
LISTCNTL	Program output listing controls
LISTLIB GLOSSARY	Catalog content glossary listing controls
LISTLIB NAMES	Catalog content names listing controls
MULTILIB	Multiple M4LIB ordering parameters
OVERRIDE	DD Name override parameters
OWNCODE	Own Code parameters
RETRIEVE	Catalog content retrieval controls
ROUTE	Report routing parameters and controls
TRACK	Catalog item tracking parameters
WORK	Working storage parameters

The following rules define what statements may be used for each of the various types of runs:

- Dictionary maintenance runs may only use the CONTROL, COPY, LISTLIB, and TRACK commands.
- File processing runs may use all commands except the LISTLIB and RETRIEVE commands.
- Report generation runs (step 3 of a 3-step run) may only use the CHECKPOINT, CONTROL, COPY, and FILE REPORT commands.
- Source statement retrieval runs may only use the CONTROL and RETRIEVE commands.

ARRAY Command

```
ARRAY [QUALIFIER] qualifier-char,
      NAME definition-name,
      [OVERDEFINES qualifier-char]
```

The ARRAY command is used to define internal arrays. Multiple ARRAY commands may be used to identify multiple arrays in an application.

QUALIFIER <i>qualifier-char</i> QUAL	Specifies a unique identifier for the array. Valid <i>qualifier-char</i> values are A, B, E, G, H, J, K, M, Q, or 1-9. This array identifier also becomes the qualifier for any field names within the array. If a numeric qualifier is assigned (1-9), a FILE CORDn may not be specified that uses the same numeric designation.
NAME <i>definition-name</i>	Identifies the array definition that describes the elements within the array.
OVERDEFINES <i>qualifier-char</i> OVER	Identifies the target array that this array definition over-defines. The over-defined array may either be another array defined with an ARRAY command or an array defined with a FILE CORDn command. The data-cell size times the number of columns of this array must equal the data-cell size times the number of columns in the over-defined array. The size of a row must be equal in both arrays and the number of rows in this array must be less than or equal to the number of rows in the target array.

Examples

```
ARRAY QUAL A NAME ARRAY1
```

Allocate an array using the array definition ARRAY and assign it a qualifier of A.

```
ARRAY B NAME ARRAY2 OVERDEFINES A
```

Allocate an array with qualifier B that over-defines array A using array definition ARRAY2.

CATALOG Command

```
CATALOG {SAVE [GROUP group-name] [REQUESTS request-name ...] |  
        INSERT REQUEST request-name INTO group-name [AFTER request-name] |  
        DELETE [GROUP group-name] [REQUESTS request-name ...] |  
        REPLACE REQUESTS request-name ... |  
        DUMP {ALL | ITEMS item-names ... } |  
        LIST}
```

The CATALOG command is used to specify cataloging operations for requests or request groups. Requests or request groups are inserted into the application code using the procedural INCLUDE command. Multiple CATALOG commands may be present in a single application.

SAVE	Save a request or group in the common library. When the GROUP keyword is omitted but the REQUEST keyword is present, the identified requests are saved as individual requests (not contained in any group). When the REQUEST keyword is omitted but the GROUP keyword is present, all requests in the run are saved as requests within the identified group. If both the GROUP and REQUEST keywords are present, the identified requests are saved within the identified group.
INSERT	Insert a request into an existing group within the common library. The REQUEST and INTO keywords are required. If the AFTER keyword is omitted, the request is inserted as the first request in the group.
DELETE	Delete a request or group from the common library or delete a request from within a group. If both the REQUEST and GROUP keywords are present, the requests are deleted from the designated group. If the GROUP keyword only is present, the group is deleted. If the REQUEST keyword only is present, the individual requests (not contained in any group) are deleted.
REPLACE	Replace a request in all groups in which it occurs within the common library.
DUMP	List the source statements contained in a request or group within the common library. The ALL and ITEM keywords (see below) are used to specify the scope of the dump operation.

LIST	List the names of all requests and groups within the common library. Only one list operation is performed even when more than one CATALOG LIST command is specified.
GROUP <i>group-name</i>	Identifies the name of the group to which the corresponding operation applies.
REQUESTS <i>request-name</i> REQUEST	Identifies the name of one or more requests to which the corresponding operation applies.
INTO <i>group-name</i>	Identifies the name of a group into which a request is inserted.
AFTER <i>request-name</i>	Identifies the name of a request after which the inserted request is to be placed.
ALL	Indicates that the source statements of all requests and groups within the common library are to be listed.
ITEMS <i>item-names</i> ITEM	Identifies the names of requests and/or groups that are to be included in the source statement list process.

Examples

```
CATALOG LIST
```

List the names of all requests and request groups that are cataloged in the common library.

```
CATALOG SAVE GROUP XYZ, REQUESTS ABC DEF
```

Save requests ABC and DEF into a group called XYZ.

```
CATALOG DUMP ITEMS XYZ ABC DEF
```

Display the source statements in items XYZ, ABC, and DEF in the common library.

CHECKPOINT Command

```
CHECKPOINT [COUNT number],  
           [FILE ddname],  
           [TIME number {MINUTES | SECONDS}],  
           [EOV ddname],  
           [OPERATOR],  
           [PREFIX id-prefix],  
           [COMMITONLY],  
           [ALTERNATING]
```

The CHECKPOINT command is used to specify that checkpoint operations are to occur during the execution of the application. Only one CHECKPOINT command may be present in an application but more than one of the available event triggers may be specified. If the application is invoked via the IMS region controller (program name DFSRRC00), IMS checkpoints will be taken. Otherwise, standard operating system checkpoints (CHKPT macro) will be taken.

COUNT <i>number</i>	Specifies a record count interval that will trigger a checkpoint operation when the records processed count reaches the specified number. Reads only of existing records or insertions of new records will increment the records processed count by 1. An update-in-place or deletion of an existing record will increment the records processed count by 2. The records processed count is initialized to 0 at the beginning of application execution and after every checkpoint event. A FILE keyword is also required whenever this keyword is present.
FILE <i>ddname</i>	Specifies the DD name of the logical file to which the COUNT keyword applies. This keyword is required whenever the COUNT keyword is present. The DD name of M4OLD is used for the master file even when there is no physical DD statement by that name in the JCL. Likewise, M4CORDn is used for coordinated files in similar circumstances.
TIME <i>number</i>	Specifies an elapsed clock time interval that will trigger a checkpoint operation. The MINUTES or SECONDS keyword may be used to specify the units for the TIME keyword. If both the MINUTES and SECONDS keywords are omitted when the TIME keyword is present, the unit of time will be assumed to be minutes.

MINUTES	Specifies that the units of the TIME keyword are in minutes. The TIME keyword must be present if the MINUTES keyword is present and the MINUTES keyword is mutually exclusive with the SECONDS keyword.
SECONDS	Specifies that the units of the TIME keyword are in seconds. The TIME keyword must be present if the SECONDS keyword is present and the SECONDS keyword is mutually exclusive with the MINUTES keyword.
EOV <i>ddname</i>	Specifies the DD name of the logical file for which an end-of-volume condition will trigger a checkpoint operation. See the FILE keyword above for the rules regarding specifying DD names.
OPERATOR	Specifies that an operator control request is to be issued and that a response by the operator will trigger a checkpoint operation.
PREFIX <i>id-prefix</i>	Specifies the checkpoint id prefix when the IMS checkpoint mechanism is used.
COMMITONLY	Specifies that a DB2 commit only will be performed when a checkpoint event is to occur. No restart is possible when this keyword is present as no log containing restart information is written. The operating system or IMS checkpoint functions will not be invoked.
ALTERNATING	Specifies that the operating system checkpoint records are to be written to alternating files. This keyword is not valid when the IMS checkpoint mechanism is used.

Examples

```
CHECKPOINT COUNT 100, FILE M4OLD
```

A checkpoint will be taken after the first 100 records have been read from the file from the master file and every 100 records thereafter.

```
CHECKPOINT TIME 5 MINUTES, COMMITONLY
```

A checkpoint will be taken every 5 minutes of elapsed time and the checkpoint function will consist only of a DB2 COMMIT operation.

COLLATE Command

```
COLLATE {[REPORTS] report-name ... [KEYLENGTH length] |  
        ALL KEYLENGTH length}
```

The COLLATE command specifies the reports that are to be collated and the length of the collating key field. An application may contain many COLLATE commands but any given report may only appear on one COLLATE command.

REPORTS *report-name* ... Specifies one or more reports that are to be collated and that become the members of this collection of reports. The *report-name* operand is a list of one or more request names (fixed-syntax object) or report names (ASL object). If the request contains more than one report, specific reports within the request may be identified by stating the request name, a colon and the specific report number(s) (Rn statement sets). For example, SALESRPT:513 indicates that only reports 1, 3, and 5 of the request SALESRPT are to be considered for this COLLATE command and that report 5 will print before reports 1 and 3. No embedded blanks are allowed in the list of report numbers.

The order of the request names and report numbers within requests (fixed-syntax objects), and the report names (ASL objects) determines the order of the final report output. The COLLATE command may be used to re-order the report output sequence.

ALL Specifies that all reports in the application are to be collated according to the data values contained in the collating key whose length is specified via the KEYLENGTH parameter. When the ALL keyword is specified, the KEYLENGTH parameter is required and only one COLLATE command is allowed in the application.

KEYLENGTH *length*
KEYLEN

Specifies the size of the collating key, beginning at the first byte of the report key, used to control the collation of report data. The length specified must not exceed the length of the report key. All report data for the reports identified with the REPORTS keyword is grouped by the values of the collating key.

When the KEYLENGTH keyword is not specified in combination with the REPORTS keyword, the effect of the COLLATE command is to print the full reports in the order listed within the REPORTS keyword parameters. No collation of report subsets from the different reports will take place.

Example

```
COLLATE REPORTS REP1 REP2 KEYLENGTH 1
```

REP1 and REP2 will be collated according to the collating key which is one byte in length. REP1 will print before REP2.

CONTROL Command

```
CONTROL [NAME run-name],  
  
    [DELIMITER 'x'],  
    [{SCANONLY | SAMPLE | MAPDECODE}],  
    [{TERM | CONTINUE}],  
    [SORT {INTERNAL | EXTERNAL | NONE}, [SUMMARIZE]],  
    [{NOLIST | NOSOURCE | LISTGEN}],  
    [AMODE {31 | 24}],  
    [ABEND],  
    [GRANDSUM],  
    [CORDONLY],  
    [GETMAIN nnnnK],  
    [SORTSIZE nnnnK],  
    [REPTSIZE nnnnK],  
    [FREESIZE nnnnK],  
    [DB2 subsystem-id plan-name],  
    [SQLID authorization-id],  
    [EXPLAIN query-number],  
    [{SYSDATE mmddy | SYSDATE4 yyyyymmdd}],  
    [DECMSG {YES | NO}],  
    [PROMSG {YES | NO}],  
    [RPTMSG {YES | NO}]
```

The CONTROL command is used to specify application-related parameters and controls. This command must be the first command in an application. Even though no keywords are specified, the command must still be present.

NAME <i>run-name</i>	Specifies the name of the application. If omitted, a default name will be used.
DELIMITER 'x'	Specifies the delimiter character to be used for this application. If omitted, the default delimiter specified at installation time in M4PARAMS will be used. The operand must be a single character enclosed in single-quotes (apostrophes). The delimiter may be any character except an underscore (_) or tilde (~). The delimiter character is used in various places to delimit strings.

SCANONLY	Specifies that this invocation of VISION:Builder is a scan-only run. The syntactical correctness of all source statements will be verified, any cataloged request maintenance will be performed (see CATALOG command), and the cross-validation of application functions will be completed. No file/database processing will be done. This keyword is mutually exclusive with the SAMPLE, MAPDECODE, TERM, and CONTINUE keywords. Note that, if the application references DB2 databases, a connection to DB2 will be required in order to fully verify the correctness of any user coded or automatically derived SQL statements.
SAMPLE	<p>Specifies that in addition to the scan-only functions described above, sample reports will be generated for each valid report specification in the application. Sample reports allow you to check the layout and appearance of a report without actually preparing test data. This keyword is mutually exclusive with the SCANONLY, TERM, and CONTINUE keywords.</p> <p>If this keyword is specified in combination with the MAPCODE keyword, sample reports for any mapping requests will be generated.</p>
MAPDECODE	<p>Specifies that this invocation of VISION:Builder is for the express purpose of validating mapping requests and performing the related cataloged request maintenance functions. No file/database processing will be done. This keyword is mutually exclusive with the SCANONLY, TERM, and CONTINUE keywords.</p> <p>If the SAMPLE keyword is specified in combination with this keyword, sample reports for the mapping requests will be generated.</p>
TERM	Specifies that the application run should be terminated before any file/database processing operations occur if any syntax or cross-validation errors are detected. This keyword is mutually exclusive with the SCANONLY, SAMPLE, MAPDECODE, and CONTINUE keywords.

CONTINUE

Specifies that file/database processing should be performed ignoring any requests that are invalid and that processing should continue after a type 4 message is issued. A type 4 message indicates some sort of file/database condition that prevents a program function to complete normally. One such condition would be an attempt to insert a new record while a record with the identical key already exists in the database. This keyword should be used with caution because incomplete and even invalid results may be produced. Usually, this keyword would be specified only when some error condition is expected but that the user is willing to tolerate. The program listing should be examined carefully to determine what functions might not have completed successfully. This keyword is mutually exclusive with SCANONLY, SAMPLE, MAPDECODE, and TERM. If the SCANONLY, SAMPLE, MAPDECODE, TERM, and CONTINUE keywords are all omitted, any invalid requests are ignored but the run will be terminated whenever a type 4 error is encountered.

SORT

Specifies the sort control parameters for the application. This keyword must be followed by one of three keywords as follows:

- INTERNAL -- Specifies that sorting of the selected report data is to be performed internally in a single-step process and the report output produced as part of this process.
- EXTERNAL -- Specifies that sorting of the selected report data will be performed in a separate job step. Report output must be produced in a subsequent report generation step.
- NONE -- Specifies that no sorting of selected report data is required and that any report output will be produced as part of a single job step. This option is not valid if the application contains specifications for two or more reports.

SUMMARIZE	Specifies that for SORT EXTERNAL, the sort process is to optimize record I/O by taking preliminary summaries for numeric fields in the input detail records and producing only summarized output records. This optimization is effective for summary only reports that request TOTAL, CUM, PERCENT, or RATIO summaries on numeric fields. There can be no more than 40 summary and control fields in the report, and the length of the data record on the report file cannot exceed 400 bytes. This keyword is invalid if SORT EXTERNAL is not also specified.
NOLIST	Specifies that no listing of the application source statements is to be produced. In addition, any information-only or warning messages, the banner page, and Program Analyzer output will be suppressed. This keyword is mutually exclusive with the NOSOURCE and LISTGEN keywords.
NOSOURCE	Specifies that listing of the application source statements and the banner page is to be suppressed. This keyword is mutually exclusive with the NOLIST and LISTGEN keywords.
LISTGEN	Specifies that a complete listing of all source statements is to be produced along with the fixed-syntax equivalent statements of any ASL statements present in the application. This keyword may be useful for users familiar with the fixed-syntax statements while they are transitioning to using ASL in place of the fixed-syntax statements. This keyword is mutually exclusive with the NOLIST and NOSOURCE keywords. If neither the NOLIST, NOSOURCE, nor LISTGEN keywords are present, all source statements (fixed-syntax or ASL) will be listed as is along with any information or diagnostic messages, a banner page will be output, and Program Analyzer output will be produced.

AMODE	Specifies the addressing mode that is to be used during the file/database processing stage of the application execution. Valid mode specifications are 24 or 31. A value of 24 indicates that only processor storage below the 16-meg line will be utilized by the application. A value of 31 indicates that processor storage above the 16-meg line may be used for file buffers, table-lookup tables, and miscellaneous control blocks. If the AMODE keyword is omitted, the default value specified at installation time in M4PARAMS will be used. AMODE 24 may be required if external programs are called that can not access processor storage above the 16-meg line. Otherwise, AMODE 31 is the preferred specification.
ABEND	Specifies that, if the application run encounters any error condition, the job step will be terminated with an operating system abend rather than just setting a non-zero job-step condition code.
GRANDSUM	Specifies that grand summaries are to be automatically generated for each report in the application. If this keyword is omitted, the default value specified at installation time in M4PARAMS will be used.
CORDONLY	Specifies that the coordination follow-up pass should be bypassed. This means that the execution of coordinated requests will be bypassed when the ECORD flag is equal to all Xs regardless of whether or not coordination has taken place.
GETMAIN nnnnK	Specifies the size of working storage that VISION:Builder will request. The value of nnnn may be a number between 1 and 8192 and K indicates that the value is a multiple of 1024. If this keyword is omitted, the default value specified at installation time in M4PARAMS will be used.

SORTSIZE nnnnK Specifies the amount of storage available to the sort program in a single-step application (see SORT INTERNAL). The value of nnnn may be a number between 1 and 8192 and K indicates that the value is a multiple of 1024. If the keyword is omitted, the default value specified at installation time in M4PARAMS will be used.

REPTSIZE nnnnK Specifies the amount of storage available to the report generation process in a single-step application (see SORT INTERNAL or SORT NONE). The value of nnnn may be a number between 1 and 8192 and K indicates that the value is a multiple of 1024. If the keyword is omitted, the default value specified at installation time in M4PARAMS will be used.

FREESIZE nnnnK Specifies the amount of storage to be reserved for the database manager or external programs during database operations. The value of nnnn may be a number between 1 and 1024 and K indicates that the value is a multiple of 1024. If the keyword is omitted, a default value based upon the total storage available to the application will be reserved for database manager functions.

DB2 *subsystem-id plan-name*

The presence of this keyword specifies that a CALL ATTACH is to be used to connect to DB2 for this application. When this keyword is present, two operands must be specified following the keyword. The first operand of the DB2 keyword specifies the DB2 subsystem identifier and may be no more than 4 characters in length. The second operand of the DB2 keyword specifies the plan name to be used to connect to DB2 and may be up to 8 characters in length. If this keyword is omitted, CALL ATTACH will not be used to connect to DB2. Either IMS or TSO attach will be used based upon the following criteria:

1. IMS Attach will be used when the application is running under the IMS region controller
2. TSO Attach will be used in all other circumstances

SQLID <i>authorizaton-id</i>	Specifies the DB2 authorization-id that is to be used for this application execution. If omitted, the authorization id will be obtained by DB2 via the normal procedures (JCL, TSO, program exits, etc.) This keyword may be required to override the default authorization id in order to allow access to restricted DB2 resources.
EXPLAIN <i>query-number</i>	Specifies that DB2 is to insert performance information into the authorization-id.PLAN_TABLE for each EXPLAINable DB2 statement within the application using the query-number value as a key. The query number is incremented by one after each EXPLAINable DB2 statement. The applicable DB2 statements may be provided by the user via the SQL keyword on a FILE command or derived automatically by VISION:Builder from a relational database file definition. If this keyword is omitted, no DB2 performance information is captured.
SYSDATE <i>mmddy</i>	Specifies that the <i>mmddy</i> value is to be used as the date to which the date flags will be initialized and as the report date when requested. The supplied value must be in the same format specified for the TODAY flag in M4PARAMS; e.g. if the TODAY flag format in M4PARAMS is YYMMDD, then the SYSDATE value must be specified in the same format. When the year (yy) is specified as 00 through 89, the century will be interpreted as 20. Otherwise, the century will be interpreted as 19. If this keyword is omitted, the current date obtained from the operating system will be used. This keyword is mutually exclusive with the SYSDATE4 keyword.
SYSDATE4 <i>yyyymmdd</i>	Specifies that the <i>yyyymmdd</i> value is to be used as the date to which the date flags will be initialized and as the report date when requested. The supplied value must be specified exactly as shown with a 4-digit year. This keyword is mutually exclusive with the SYSDATE keyword.

DECMSGS	Specifies whether decode time information-only and warning messages are to be displayed or not. When the keyword is followed by YES, the messages will be displayed. When the keyword is followed by NO, the messages will not be displayed. If the DECMSGS keyword is omitted, the default setting specified at installation time in M4PARAMS will be used.
PROMSGS	Specifies whether processing time information-only and warning messages are to be displayed or not. When the keyword is followed by YES, the messages will be displayed. When the keyword is followed by NO, the messages will not be displayed. If the PROMSGS keyword is omitted, the default setting specified at installation time in M4PARAMS will be used.
RPTMSGGS	Specifies whether report time information-only and warning messages are to be displayed or not. When the keyword is followed by YES, the messages will be displayed. When the keyword is followed by NO, the messages will not be displayed. If the DECMSGS keyword is omitted, the default setting specified at installation time in M4PARAMS will be used.

Examples

```
CONTROL DB2 D61A MYPLAN, ABEND, LISTGEN
```

Use the CALL ATTACH facility to connect to DB2, abend the job step if any errors are detected, and list the source statements along with the fixed-syntax equivalent statements for any ASL source statements.

```
CONTROL GETMAIN 6144K, SORT EXTERNAL, GRANDSUM, DECMSGS NO
```

Request allocation of 6144K as working storage for the application, do not invoke the sort program internally (sorting will be performed in an external job-step), produce grand summaries for every report containing summary specifications, and suppress the decode-time information only and warning messages.

COPY Command

```
COPY [DDNAME] ddname [ member-name ] ,  
      [FIXED]
```

The COPY command may be included anywhere in the run control section. The commands in the referenced file are copied into the source code input stream at the point of the COPY command, and processed in an identical manner as if the commands had been in-stream. If the FIXED keyword is specified on the COPY command, the commands are assumed to be VISION:Builder fixed-syntax statements and they are processed as such. Copied members from a COPY without the FIXED keyword may in turn contain COPY commands. A COPY command with the FIXED keyword prior to a PROC command or a ;;ENDASL comment statement signals the end of the run control section of the application. This means that no run control commands may appear following a COPY command with the FIXED keyword. Typically, a COPY command with the FIXED keyword would either include definitions from a Definition Library (created using the Workbench for ISPF) or fixed-syntax statements for requests.

Examples

```
COPY COPYLIB (PROXZY)
```

Copy the statements in member PROXZY in the dataset associated with DD name COPYLIB into the input stream as ASL statements.

```
COPY FILEA
```

Copy the statements in the dataset associated with DD name FILEA into the input stream as ASL statements.

```
COPY DEFLIB (MYDEF) FIXED
```

Copy the statements in member MYDEF associated with DD name DEFLIB into the input stream as fixed-syntax statements. This form of the COPY command is an easy way to insert definitions prepared with VISION:Workbench for ISPF into the input stream for cataloging.

DEBUG Command

```
DEBUG [CLEAR] ,  
      [DUMP] ,  
      [LONGNAMES] ,  
      [TRACE COMPCODE + IMSCALLA + MAPPING + SQLCALL]
```

The DEBUG command is used to specify special controls and debugging output that may be useful for debugging complex applications.

CLEAR	Specifies that any processor storage that is acquired should be cleared (set to binary zeros) before it is used.
DUMP	Specifies that if an application abend is forced by VISION:Builder (see the CONTROL command ABEND keyword), the dump option should be specified on the abend control.
LONGNAMES	Specifies that a cross-reference of short field names to long field names should be output with the program listing. This may be useful for debugging when diagnostic output refers only to the short field name.
TRACE <i>option</i> ...	Specifies one or tracing options for the application. The valid options are: <ul style="list-style-type: none">■ COMPCODE -- Specifies that the code compiled for an application be included in the program listing. This information will not be very meaningful for users but may assist CA support personnel in problem diagnosis.■ IMSCALLA -- Specifies that DL/I calls for IMS databases are to be traced. This option should only be used in controlled debugging situations in order to avoid large amounts of output and excessive processing overhead.■ MAPPING -- Specifies that calls to the Generalized Database Interface functions are to be traced. This option should only be used in controlled debugging situations in order to avoid large amounts of output and excessive processing overhead.■ SQLCALL -- Specifies that calls related to any DB2 databases are to be traced. This option should only be used in controlled debugging situations in order to avoid large amounts of output and excessive processing overhead.

Examples

```
DEBUG DUMP, TRACE IMSCALLA
```

Include the dump option on anyabend and trace all DL/I calls.

```
DEBUG TRACE COMPCODE SQLCALL
```

Output a listing of the compiled code and trace all SQL statement calls.

DOCUMENT Command

```
DOCUMENT [CONVMMSG],  
        [XREF],  
        [EXECTRACE],  
        [MAXLINES number]
```

The DOCUMENT command is an optional command used to generate additional listing information to enhance program documentation and assist in debugging your VISION:Builder programs. Refer to the VISION:Builder Reference Guide for a complete description of the Program Analyzer (PAL) facility.

CONVMMSG	Output messages detailing the numeric conversions that are required to perform the program operations.
XREF	Extract data from which cross-reference reports may be created. The M4PAOUT file will be used to collect the cross-reference data and the reports must be generated in a subsequent process.
EXECTRACE	Extract data for the execution trace reports. The reports contain information on standard flag settings and GDBI mapping request events. The M4PAOUT file will be used to collect the cross-reference data and the reports must be generated in a subsequent process.
MAXLINES <i>number</i>	Specifies the maximum number of lines of execution trace report data to produce. If the number is appended with a K, then the number is assumed to be a multiple of 1024.

Examples

```
DOCUMENT CONVMMSG, EXECTRACE, MAXLINES 5000
```

Produce additional application documentation in the form of numeric conversion details and an execution trace report. Limit the execution trace report to 5000 lines.

FILE AUDIT Command

FILE AUDIT [DDNAME *ddname*]

The FILE AUDIT command is used to specify that an audit file (M4AUDIT) be created containing all records deleted from the master file. When this command is omitted, no audit file is created.

DDNAME *ddname* Specifies the DD name to be used for this file. If omitted, a DD name of M4AUDIT will be used.

Examples

```
FILE AUDIT
```

An audit file containing all records deleted from the master file is to be created by this application.

FILE CORDn Command

```
FILE CORDn {[NAME] definition-name | SQL "sql-select-statement"},
    [MOSAIC],
    [CHKORDER],
    [ARRAY],
    [{SEGMENT segment-name WHERE "sql-where-clause" ... |
     SEGMENT segment-name SSA "pre-selection-ssa" ... }],
    [{PASSWORD password | AUTHID authorization-id}],
    [IOPLUGIN module-name],
    [DDNAME ddname]
```

Additional keywords for Indexed Direct Coordination (ICF):

```
[DIRECT BY q.fldname]
```

Additional keywords for Standard Coordination:

```
[STANDARD],
[ALLRECS | MATCHONLY],
[KEYNAME field-name ...]
```

Additional keywords for Chained Coordination:

```
[CHAIN TO qualifier],
[KEYNAME field-name ...]
```

Additional keywords for User Read files:

```
[USERREAD],
[GENERIC field-name]
```

The FILE CORDn command is used to specify that the corresponding file is used as a coordinated file within the application or as an external array. Up to 9 FILE CORDn commands may be specified where n is a number from 1 through 9. This number becomes the qualifier for referencing fields within the file and is mutually exclusive with any qualifiers designated with the ARRAY command.

Coordinated files in VISION:Builder are secondary input files that may be accessed either sequentially, directly by key, or a combination thereof. Sequentially accessed coordinated files are usually synchronized with the master file but may also be synchronized with other coordinated files. A coordinated file accessed directly by key, also referred to as an indexed coordinated file (ICF), uses a field outside of the file to determine which records in the coordinated file to access. Additionally, a CORDn file may be specified as a "user read" file in which case all access is controlled procedurally by the DO FORALL command or FIND function. See [Chapter 4, Procedural Command Group](#).

NAME	Specifies the name of the file definition related to this coordinated file. The file definition must have either been previously cataloged in the common library or included in-stream with a COPY command. This keyword is mutually exclusive with the SQL keyword.
SQL " <i>sql-select-statement</i> "	Specifies a DB2 SELECT statement to be used to access the DB2 table rows. The SELECT statement text must be enclosed within quotation marks (") and may consist of multiple segments continued over a number of lines, each enclosed with quotation marks ("). Any valid DB2 SELECT statement may be specified. A file definition will be automatically derived from the SELECT statement and a glossary for the file definition will be displayed unless suppressed by the LISTCNTL INGLOSS NO command. The DB2 column names become the long field names in the file definition and these long names should be used in other ASL statements to reference the individual columns (fields). This keyword is mutually exclusive with the NAME keyword.

MOSAIC	Specifies that memory optimized processing is to be employed while accessing this file. Memory optimized processing is only meaningful for hierarchically structured databases such as IMS or when a collection of DB2 tables is defined to VISION:Builder as a hierarchical structure. The specification means that, at most, only one occurrence of each segment type in the hierarchy is retained in processor storage at any one time. When this keyword is omitted, all occurrences of each segment type subordinate to a given root segment will be retrieved and retained in processor storage for the duration of the processing cycle for the record. When the MOSAIC optimization is used, care should be taken in the program logic so that the hierarchic structure is traversed in a logical fashion. Otherwise, a particular occurrence of a segment may be retrieved multiple times increasing the application processing time. Although the purpose of this keyword is to specify a tradeoff of database access calls in favor of reduced memory requirements, judicious use of this feature along with careful coding techniques may result in optimizing both processor storage and database access calls. The procedural FIND FIRST and FIND LAST functions may only be used with IMS databases when this keyword is present.
CHKORDER	Specifies that the order of incoming segments is to be checked according to the segment key specifications in the associated file definition. This specification is ignored if the MOSAIC keyword is present.
ARRAY	Specifies that this file is an external array and that the definition identified via the NAME keyword is an Array Definition rather than a File Definition.
SEGMENT <i>segment-name</i>	Specifies the segment name in the file definition to which the following WHERE or SSA keywords are to be associated. The SEGMENT plus WHERE or SEGMENT plus SSA keywords must always be specified in pairs. The SEGMENT keyword is mutually exclusive with the SQL keyword.

WHERE " <i>sql-where-clause</i> "	<p>Specifies additional DB2 SELECT statement qualification that is to be appended to any qualification that is automatically generated by VISION:Builder based upon the file definition and file usage controls. The qualification is expressed as a logical expression consistent with the DB2 WHERE clause syntax. The expression may consist of multiple text segments enclosed in quotation marks (") each continued over multiple lines. A SEGMENT keyword must precede this keyword, and the WHERE keyword is mutually exclusive with the SQL keyword.</p> <p>Multiple SEGMENT/WHERE keyword pairs may be specified -- one set for each segment defined in the file definition.</p>
SSA " <i>pre-selection-ssa</i> "	<p>Specifies a DL/I SSA expression that qualifies the retrieval of segment occurrences. The qualification is expressed as a logical expression consistent with the rules for coding DL/I SSA expressions. The expression may consist of multiple text segments enclosed in quotation marks (") each continued over multiple lines. A SEGMENT keyword must precede this keyword, and the SSA keyword is mutually exclusive with the SQL keyword.</p> <p>Multiple SEGMENT/SSA keyword pairs may be specified -- one set for each segment defined in the file definition.</p>
PASSWORD <i>password</i>	<p>Specifies the password for password protected VSAM files. If no password is required, this keyword should be omitted.</p>
AUTHID <i>authorization-id</i>	<p>Specifies the DB2 creator id/authorization id for the table when the id is not specified in the file definition. If the coordinated file is not a DB2 database, this keyword should be omitted.</p>
IOPLUGIN <i>module-name</i>	<p>Specifies the module name that should be called in place of the standard VISION:Builder module to perform the I/O functions related to this file.</p>

DDNAME *ddname* Specifies the DD name to be used for this file. This keyword is ignored if the file is associated with a DB2 or IMS database or the IOPLUGIN keyword is present. If this keyword is omitted for files that are not a DB2 or IMS database, the DD name of M4CORDn is used where n is the number that corresponds to n in CORDn.

Additional keywords for Indexed Direct (ICF) coordination:

DIRECT BY *q.flname* Specifies the name of the field that controls which records are accessed from the file.

- The field can be in any other file, temporary field, working storage field, or linkage section field.
- If the field properties are different from the properties defined for the key field in this file, the field is converted to the properties of the defined key field.
- An illegal conversion is treated as a "record missing" or uncoordinated condition.
- If the SQL keyword is used to specify the DB2 SELECT statement, the SELECT statement must include an appropriate SELECT statement WHERE clause that qualifies the selected rows from the DB2 table in accordance with the coordinating field. Typically this would result in the use of the coordinating field as a host variable in the SELECT statement WHERE clause.
- If this file is defined as a relational file whose root segment contains a key that spans multiple columns, this field must be a character type field and contain sub-fields that correspond to the individual columns that constitute the key. A WHERE keyword must be specified that uses host variables that qualify the DB2 SELECT for each column. See the VISION:Builder Reference Guide for more information.

Additional keywords for standard coordination:

STANDARD	Specifies that this file is accessed sequentially and coordinated in the standard fashion. That is, the keys of the coordinated file are matched against the keys of the master file and the file movement is synchronized accordingly. The master file must also be accessed sequentially when this keyword is specified. Standard coordination is the default when DIRECT, CHAIN, or USERREAD is not specified.
ALLRECS	Specifies that all records in this sequentially coordinated file are to be made available to the application, not just the ones whose keys match. The ECORD flag may be used to determine the match status of each record.
MATCHONLY	Specifies that only records whose key(s) match the master file are to be made available to the application. This is the default if neither ALLRECS nor MATCHONLY is specified.
KEYNAME <i>field-name ...</i>	Specifies the name(s) of the fields to be used as key fields whose values will be matched against the key fields of the master file to determine the coordination state. Up to 3 fields may be specified. If this keyword is omitted but the NAME keyword is present, the key fields defined in the file definition will be used as the match fields. If this keyword is omitted but the SQL keyword is present, the first eligible column in the DB2 SELECT statement will become the only key field. Columns that are not eligible to become key fields are VARCHAR or GRAPHIC type columns, or CHAR type columns longer than 255 characters.

Additional keywords for chained coordination:

CHAIN TO <i>qualifier</i>	Specifies the qualifier (M or 1 through 9) of the file to which this file's movements are to be synchronized. If qualifier is M, the file movement is synchronized with the master file movement. If qualifier is 1-9, the file movement is synchronized with the corresponding CORDn file. All files in the "chain" sequence must be accessed sequentially. This specification may be used to simulate a hierarchy with a group of files.
---------------------------	--

KEYNAME *field-name ...* Specifies the name(s) of the fields to be used as coordinated file key fields whose values will be matched against the key fields of the master file to determine the coordination state. Up to 3 fields may be specified. If this keyword is omitted, the key fields defined in the file definition will be used as the match fields.

Additional keywords for "user read" files:

USERREAD Specifies that this file's movement will be controlled procedurally via the DO FORALL command or FIND function. See [Chapter 4, Procedural Command Group](#)

GENERIC *field-name* Specifies the name of the field that is a subset of the full key field defined for this file. The field must begin at the same position in the record as the full key field does but this field's length may be less than or equal to the full key field length. This specification in combination with the FIND command may be used to limit access to a range of records whose partial keys match a specific value.

Examples

```
FILE CORD1 NAME DEFY
```

Coordinated file M4CORD1 is a standard sequentially coordinated file in this application, and its file definition name is DEFY.

```
FILE CORD2 NAME DEFY DIRECT BY T.TEMP1
```

Coordinated file M4CORD2 is an indexed coordinated file (ICF) in this application, field T.TEMP1 is the coordinating field, and DEFY is the file definition name.

```
FILE CORD3 KEYNAME WORKDEPT,  
SQL "SELECT EMPNO, WORKDEPT, FIRSTNME, MIDINIT, LASTNAME, ",  
    "BIRTHDATE, SALARY FROM DSN8610.EMP ",  
    "ORDER BY WORKDEPT"
```

Coordinated file M4CORD3 is a DB2 database that is accessed sequentially for standard coordination using the specified DB2 SELECT statement and whose key field is WORKDEPT.

```
FILE CORD4 ARRAY NAME ARRAY9
```

Coordinated file M4CORD4 is an external array, and its array definition name is ARRAY9.

```
FILE CORD5 DIRECT BY WORKDEPT,  
SQL "SELECT DEPTNO, DEPTNAME FROM DSN8610.DEPT ",  
"WHERE DEPTNO = :WORKDEPT"
```

Coordinated file M4CORD5 is a DB2 table that is accessed directly, and the coordinating field is the master file field WORKDEPT.

```
FILE CORD6 USER READ, GENERIC WORKDEPT,  
SQL "SELECT WORKDEPT, EMPNO, FIRSTNME, MIDINIT, LASTNAME ",  
"FROM DSN8610.EMP ",  
"WHERE WORKDEPT = :O.DEPTNO"
```

Coordinated file M4CORD6 is a user read coordinated file using a DB2 table, and WORKDEPT is used as the generic key in the table.

FILE MASTER Command

There are three variations of the FILE MASTER command; one each for file input/update, file output, or file in memory only.

Variation for input/update:

```
FILE MASTER {INPUT | UPDATE},

    {[NAME] definition-name | SQL "sql-select-statement"},
    [ACCESS {SEQUENTIAL | DIRECT | PHYSICAL}],
    [KEYS {UNIQUE | EQUAL | NONE}],
    [STARTKEY 'key-value'],
    [ENDKEY 'key-value'],
    [KEYNAME field-name ...],
    [MOSAIC],
    [CHKORDER],
    [ONEBUFFER],
    [{SEGMENT segment-name WHERE "sql-where-clause" ... |
     SEGMENT segment-name SSA "pre-selection-ssa" ... }],
    [{PASSWORD password | AUTHID authorization-id}],
    [IOPLUGIN module-name],
    [DDNAME ddname]
```

Variation for output:

```
FILE MASTER OUTPUT,

    [NAME] definition-name,
    [{PASSWORD password | AUTHID authorization-id}],
    [IOPLUGIN module-name],
    [DDNAME ddname]
```

Variation for in-memory only:

```
FILE MASTER DUMMY,

    [NAME] definition-name
```

At least one FILE MASTER command must be present in every application that performs any file processing functions. The following rules apply to the use of the FILE MASTER command:

- Both FILE MASTER INPUT and FILE MASTER OUTPUT may be present in the same application.
- FILE MASTER OUTPUT is not allowed when FILE MASTER UPDATE is present.
- FILE MASTER DUMMY is not allowed when any other FILE MASTER commands are present.
- When FILE MASTER DUMMY is present, FILE TRAN must also be present.

INPUT	Specifies that an input (old) master file is present in this application.
UPDATE	Specifies that an input master file with update-in-place processing is present in this application.
OUTPUT	Specifies that an output (new) master file is present in this application.
DUMMY	Specifies that there is no external master file for this application, only an in-memory buffer in which logical records are constructed from transaction file data.
NAME	Specifies the name of the file definition related to the master file. The file definition must have either been previously cataloged in the common library or included in-stream with a COPY command. This keyword must be omitted if the SQL keyword is present.
SQL " <i>sql-select-statement</i> "	Specifies a DB2 SELECT statement to be used to access the DB2 table rows. The SELECT statement text must be enclosed within quotation marks (") and may consist of multiple segments continued over a number of lines, each enclosed with quotation marks ("). Any valid DB2 SELECT statement may be specified. A file definition will be automatically derived from the SELECT statement and a glossary for the file definition will be displayed unless suppressed by the LISTCNTL INGLOSS NO command. The DB2 column names become the long field names in the file definition and these long names should be used in other ASL statements to reference the individual columns (fields).

ACCESS *access-type*

Specifies how the master file will be accessed.

- **SEQUENTIAL** -- Specifies that the file will be accessed sequentially. For key sequenced VSAM files, this means that the file will be accessed in ascending sequence by key. This is the default if the **ACCESS** keyword is omitted.
- **DIRECT** -- Specifies that the file will be accessed directly by key.
- **PHYSICAL** -- Specifies that the file will be accessed in physical sequential order. For key sequenced VSAM files, this means that the order of records received may not be in ascending sequence by key.

KEYS *key-control*

Specifies how the master file record keys are to be checked.

KEYS EQUAL or **KEYS UNIQUE** is not allowed when **ACCESS PHYSICAL** is specified.

- **UNIQUE** -- Specifies that the master file record keys are unique and in ascending sequence. This is the default if the **KEYS** keyword is omitted and **ACCESS PHYSICAL** is not specified.
- **EQUAL** -- Specifies that the master file may contain records with duplicate keys but that otherwise the keys will be in ascending sequence.
- **NONE** -- Specifies that the master file records do not have any verifiable sequence and that no key sequence should be expected or checking performed. This specification is assumed for an IMS HDAM file that is accessed sequentially. If this specification is present, the **FILE TRAN** command is not permitted.

STARTKEY ' <i>key-value</i> '	Specifies that access to the master file should be limited to only those records whose key value is equal to or greater than the specified value. Conversely, this keyword specifies that all records whose key value is less than the specified value will be skipped. The key value may not be more than 16 characters in length and will be converted to the properties of the defined key field. This keyword must be omitted if KEYS NONE is specified.
ENDKEY ' <i>key-value</i> '	Specifies that access to the master file should be limited to only those records whose key value is less than or equal to the specified value. Conversely, this keyword specifies that all records whose key value is greater than the specified value will be skipped. The key value may not be more than 16 characters in length and will be converted to the properties of the defined key field. This keyword must be omitted if KEYS NONE is specified.
KEYNAME <i>field-name ...</i>	Specifies the name(s) of the fields to be used as master file key fields for this application. Up to 3 field names may be specified. If this keyword is omitted, the key fields defined in the file definition will be used.

MOSAIC

Specifies that memory optimized processing is to be employed while accessing this file. Memory optimized processing is only meaningful for hierarchically structured files and means that, at most, only one occurrence of each segment type in the hierarchy is retained in processor storage at any one time. When this keyword is omitted, all occurrences of each segment type subordinate to a given root segment will be retrieved and retained in processor storage for the duration of the processing cycle for the record. When the MOSAIC optimization is used, care should be taken in the program logic so that the hierarchic structure is traversed in a logical fashion. Otherwise, a particular occurrence of a segment may be retrieved multiple times increasing the application processing time. Although the purpose of this keyword is to specify a tradeoff of database access calls in favor of reduced memory requirements, judicious use of this feature along with careful coding techniques may result in optimizing both processor storage and database access calls.

The following rules apply to the use of this keyword:

- No distinction is made among the master file qualifiers (blank, N, or O) when this keyword is present. Only one copy of the segment is used for both the old and new master logical records.
- The procedural FIND FIRST and FIND LAST functions may only be used with DL/I files when this keyword is present.
- This keyword is required when FILE MASTER UPDATE is specified and the logical file contains more than one DB2 table.

CHKORDER

Specifies that the order of incoming segments is to be checked according to the segment key specifications in the associated file definition. This specification is ignored if the MOSAIC keyword is present.

ONEBUFFER	Specifies that only a single buffer should be allocated for the master file record. Ordinarily, two buffers are allocated - one for the old master record and one for the new master record. This keyword is not permitted when the FILE TRAN or FILE MASTER UPDATE commands are present.
SEGMENT <i>segment-name</i>	Specifies the segment name in the file definition to which the following WHERE or SSA keywords are to be associated. The SEGMENT plus WHERE or SEGMENT plus SSA keywords must always be specified in pairs. The SEGMENT keyword is mutually exclusive with the SQL keyword.
WHERE " <i>sql-where-clause</i> "	<p>Specifies additional DB2 SELECT statement qualification that is to be appended to any qualification that is automatically generated by VISION:Builder based upon the file definition and file usage controls. The qualification is expressed as a logical expression consistent with the DB2 WHERE clause syntax. The expression may consist of multiple text segments enclosed in quotation marks (") each continued over multiple lines. A SEGMENT keyword must precede this keyword, and the WHERE keyword is mutually exclusive with the SQL keyword.</p> <p>Multiple SEGMENT/WHERE keyword pairs may be specified -- one set for each segment defined in the file definition.</p>
SSA " <i>pre-selection-ssa</i> "	<p>Specifies a DL/I SSA expression that qualifies the retrieval of segment occurrences. The qualification is expressed as a logical expression consistent with the rules for coding DL/I SSA expressions. The expression may consist of multiple text segments enclosed in quotation marks (") each continued over multiple lines. A SEGMENT keyword must precede this keyword, and the SSA keyword is mutually exclusive with the SQL keyword.</p> <p>Multiple SEGMENT/SSA keyword pairs may be specified -- one set for each segment defined in the file definition.</p>

PASSWORD <i>password</i>	Specifies the password for password protected VSAM files. If no password is required, this keyword should be omitted. PASSWORD is mutually exclusive with AUTHID.
AUTHID <i>authorization-id</i>	Specifies the DB2 creator id/authorization id for the table when the id is not specified in the file definition. If the master file is not a DB2 database, this keyword should be omitted. AUTHID is mutually exclusive with PASSWORD.
IOPLUGIN <i>module-name</i>	Specifies the module name that should be called in place of the standard VISION:Builder module to perform the I/O functions related to this file.
DDNAME <i>ddname</i>	Specifies the DD name to be used for this file. This keyword is ignored if the file is associated with a DB2 or IMS database or the IOPLUGIN keyword is present. If this keyword is omitted for files that are not a DB2 or IMS database, the DD name of M4OLD is used for FILE INPUT or FILE UPDATE and the DD name M4NEW is used for FILE OUTPUT.

Examples

```
FILE MASTER INPUT, NAME DEF1
```

The master file is a sequentially accessed master file whose file definition name is DEF1.

```
FILE MASTER INPUT, KEYNAME BIRTHDATE, KEYS EQUAL,  
SQL "SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, ",  
    "BIRTHDATE, SALARY FROM DSN8610.EMP ",  
    "ORDER BY BIRTHDATE"
```

This master file is a DB2 table, column name BIRTHDATE is to be used as the key, and equal keys are accepted.

```
FILE MASTER INPUT,  
SQL "SELECT X.NAME, Y.NAME, CREATOR ",  
    "FROM SYSIBM.SYSCOLUMNS X, SYSIBM.SYSTABLES Y ",  
    "WHERE X.TBNAME = Y.NAME AND CREATOR = 'REDJ004'"
```

This master file contains data that is the result of a join of two DB2 tables.

```
FILE MASTER INPUT,  
SQL "SELECT WORKDEPT, SUM(SALARY) AS TOTAL_SALARY",  
    "FROM DSN8610.EMP",  
    "GROUP BY WORKDEPT ORDER BY WORKDEPT"
```

This FILE MASTER command specifies the DB2 built-in function of SUM for the SALARY column and names the column as TOTAL_SALARY. Each row received for this SELECT will contain a department code and the total salaries for all employees in that department. The name TOTAL_SALARY must be used as the name of the column when it is referenced in other ASL statements.

```
FILE MASTER INPUT, NAME TEMPL, KEYS NONE, SEGMENT TEMPL,  
WHERE "SALARY = (SELECT MAX(SALARY) FROM DSN8610.EMP",  
      "WHERE WORKDEPT = X1.WORKDEPT)",  
      "ORDER BY SALARY DESC"
```

This FILE MASTER command specifies a WHERE condition for the DB2 table whose segment name is TEMPL as defined in file definition TEMPL. The DB2 table rows will be sorted by SALARY in descending order.

```
FILE MASTER INPUT, NAME EMPXFD,  
SEGMENT EMPLOYEE SSA "O.EMPSTAT EQ 'A'",  
SEGMENT HISTORY SSA "O.SALARY GT 30000 AND",  
                    "O.SALARY LT 50000"
```

This FILE MASTER command is for an IMS database whose file definition name is EMPXFD. Additionally, SSA qualifications are specified for segment names EMPLOYEE and HISTORY in the IMS database.

```
FILE MASTER UPDATE, ACCESS DIRECT, MOSAIC, NAME EMPXFD
```

This FILE MASTER command specifies that the master file is to be updated in place, accessed directly using the transaction file (only master file records whose key matches the transaction file records will be accessed), use MOSAIC optimization, and whose file definition name is EMPXFD.

```
FILE MASTER INPUT, NAME ALL80, IOPLUGIN M4PDSIN
```

This FILE MASTER command specifies that the I/O plugin module M4PDSIN is to be used to read the file defined by file definition ALL80. This module M4PDSIN in this example is shipped with VISION:Builder and reads all of the records for each member of a partitioned data set in order by member name. See the VISION:Builder Reference Guide for more information about this plugin module.

FILE REJECT Command

FILE REJECT [DDNAME *ddname*]

The FILE REJECT command is used to specify that a rejected transaction file (M4REJECT) be created containing all rejected transaction records. When this command is omitted, no rejected transaction file is created.

DDNAME *ddname* Specifies the DD name for this file. If the keyword is omitted, a DD name of M4REJECT will be used.

Examples

```
FILE REJECT
```

A file containing all reject transaction records is to be created by this application.

FILE REPn Command

```
FILE REPn [NAME] report-file-name,  
          [DDNAME name]
```

The FILE REPn command is used to specify that alternate report files will be created in this application. Up to 8 FILE REPn commands may be present where n may be a number from 2 through 9.

NAME <i>report-file-name</i>	Specifies a name to be assigned to this alternate report file. The FILE keyword on the procedural REPORT command refers to this name and specifies that the report data is to be written to this file. Alternate report files may be used to collect report data that will be processed at a later time or to reduce the size of files to be sorted had all of the report data been collected on the primary report file.
DDNAME <i>ddname</i>	Specifies the DD name for this file. If omitted, the DD name M4REPn will be used where n corresponds to the n in REPn.

Examples

```
FILE REP2 CUSTDATA
```

Alternate file M4REP2 is available, and its name is CUSTDATA.

FILE REPORT Command

FILE REPORT [DDNAME *ddname*]

The FILE REPORT command is used to specify that a primary report file is present in this application. If a FILE MASTER command is present, then the primary report file will be created in this application and will contain all report data not directed to an alternate report file. If a FILE MASTER command is not present in this application, then the primary report file will be an input file containing the report data to be processed in this report-processing step of a 3-step application.

DDNAME *ddname* Specifies the DD name to be used for the file. If this keyword is omitted, a DD name of M4REPO will be used for processing runs and M4REPI will be used for report generation runs.

Examples

FILE REPORT

If a FILE MASTER command is also present, this command specifies that the primary report file is available to receive selected report data. If no FILE MASTER command is present, this command specifies that this is a report generation run.

FILE SUBFn Command

```
FILE SUBFn [NAME] subfile-name,  
  
    [TABLE "authid.tablename" {CREATE | DELETE | INSERT | DROP}],  
    [TABLESPACE tablespace-name],  
    [DATABASE database-name],  
    [DELETEDSEGS],  
    [AUTODEF],  
    [{PASSWORD password},  
    [IOPUGIN module-name],  
    [DDNAME ddname]
```

The FILE SUBFn command is used to specify the presence of a subfile in this application. Subfiles are files that contain extracted data. The procedural EXTRACT FILE command is used to output data to a subfile defined via the FILE SUBFn command. Up to 10 subfiles may be specified, and the n in SUBFn may be a number from 0 to 9.

NAME <i>subfile-name</i>	Specifies a name that is to be assigned to this subfile. This name will correspond to the subfile name on the EXTRACT FILE command.
TABLE " <i>authid.tablename</i> "	Specifies the authorization id and table name when the subfile data is to be output as rows in a DB2 table. If authorization id is present, the entire string must be enclosed in quotes ("").
CREATE	Specifies that a new DB2 table is to be created and new rows inserted into this table.
DELETE	Specifies that all rows in the existing DB2 table are to be deleted before any new rows are inserted.
INSERT	Specifies that new rows are to be inserted into the existing DB2 table. Previously inserted rows are left as is.
DROP	Specifies that an existing DB2 table definition for this table is to be dropped, a new table created and new rows inserted into the table. If the existing table could not be found, a new table is still created.
TABLESPACE <i>tablespace-name</i>	Specifies the name of the tablespace into which the new table is to be created for the CREATE and DROP keywords.

DATABASE <i>database-name</i>	Specifies the database name for the new table created for the CREATE and DROP commands.
DELETEDSEGS DELSEGS	Specifies that before the subfile record is written out, any segments marked for deletion will be removed from the record. This specification is only valid if the corresponding EXTRACT FILE command for this subfile includes an ENTIRE N specification (output entire new master record).
AUTODEF	Specifies that this subfile will not be used to contain extracted data records but will instead be reserved to receive file definition statements corresponding to EXTRACT commands for which the AUTODEF keyword is present. Only one FILE SUBFn command may contain this keyword.
PASSWORD <i>password</i>	Specifies the password for password protected VSAM files. If no password is required, this keyword should be omitted.
IOPLUGIN <i>module-name</i>	Specifies the module name that should be called in place of the standard VISION:Builder module to perform the I/O functions related to this file.
DDNAME <i>ddname</i>	Specifies the DD name to be used for this file. This keyword is ignored if the file is associated with a DB2 or IMS database or the IOPLUGIN keyword is present. If this keyword is omitted for files that are not a DB2 or IMS database, the DD name of M4SUBFn is used where n corresponds to the n in SUBFn.

Examples

```
FILE SUBF1, NAME OUT1
```

Subfile output file number 1 is used in this application and assigned the name of OUT1.

```
FILE SUBF2, NAME TABLE1, CREATE, TABLE "MYID.MYTABLE",  
TABLESPACE MYSPACE
```

Subfile output file number 2 is used in this application, a new DB2 table is created to receive the output, the table name is MYID.MYTABLE, and the table resides in tablespace MYSPACE.

```
FILE SUBF0, AUTODEF
```

Subfile output file number 0 is assigned to receive the file definition statements for any procedural EXTRACT statements that include the AUTODEF keyword.

FILE TRAN Command

```
FILE TRAN [[NAME] definition-name],
          [GROUPS group-name ...],
          [CHKORDER],
          [PASSWORD password],
          [IOPLUGIN module-name],
          [DDNAME ddname]
```

The FILE TRAN command specifies that a transaction file is present in this application.

NAME <i>definition-name</i>	Specifies the name of the file definition name that defines the data field in the transaction file. This keyword is optional and is only required when procedural PROC TYPE TYPE commands are present that examine or modify fields within the transaction record.
GROUPS <i>group-name ...</i>	Specifies one or more transaction group definition names that are to be included in this application. Transaction groups define the actions to be performed against the master file for each transaction record type. The names must match the transaction group names specified for the master file used in this application. If this keyword is omitted, all transaction groups defined for the master file will be included.
CHKORDER	Specifies that the order of incoming segments is to be checked according to the segment key specifications in the associated file definition.
PASSWORD <i>password</i>	Specifies the password for password protected VSAM files. If no password is required, this keyword should be omitted.
IOPLUGIN <i>module-name</i>	Specifies the module name that should be called in place of the standard VISION:Builder module to perform the I/O functions related to this file.
DDNAME <i>ddname</i>	Specifies the DD name to be used for this file. If this keyword is omitted, a DD name of M4TRAN will be used.

Examples

```
FILE TRAN
```

A transaction file is used by this application, and all transaction group definitions that have been defined for the corresponding master file will be used.

```
FILE TRAN, NAME TRANDEF1, GROUPS GROUPA GROUPB
```

A transaction file is used by this application, the file definition named TRANDEF1 defines the transaction file fields, and only transaction group definitions named GROUPA and GROUPB for master file TRANDEF1 will be used.

LINKAGE Command

LINKAGE [**AREA**] *number*,
NAME *definition-name*

The LINKAGE command is used to specify the presence and position of parameter data passed to the application. The qualifier V is used to reference data in linkage sections.

<i>AREA number</i>	Specifies the position of this particular linkage section. The area number may be a number from 1 to 99 where 1 represents the first parameter area passed to the application, 2 the second, and so on.
<i>NAME definition-name</i>	Specifies the name of the file definition that defines the detail fields within the parameter area. When multiple linkage sections are defined, the field names within all linkage section definitions must be unique.

Examples

```
LINKAGE AREA 1, NAME LINKDEF1
```

A linkage section for parameter 1 is defined, and the file definition name for the area is LINKDEF1.

```
LINKAGE AREA 4, NAME PCB4
```

If the IMS region controller invoked this application, this linkage section would refer to the fourth PCB in the parameter list.

LISTCNTL Command

```
LISTCNTL [ALTLIST {YES | NO}],  
        [FILESUM {YES | NO}],  
        [INDEF {YES | NO}],  
        [INGLOSS {YES | NO}],  
        [INREQ {YES | NO}],  
        [CATREQ {YES | NO}],  
        [MAPREQ {YES | NO}],  
        [SQLSTAT {YES | NO}],  
        [MOSAICSTAT {YES | NO}]
```

The LISTCNTL command is used to control the program listing output or direct the default report output to M4LIST1.

ALTLIST	Specifies whether the default destination for report output is M4LIST or M4LIST1. Program listing lines will always be directed to M4LIST and report phase output will also be directed to M4LIST if the LISTCNTL command is omitted entirely or the ALTLIST keyword is omitted from the LISTCNTL command. When ALTLIST YES is specified, report phase output will be directed to M4LIST1 unless the reporting FORMAT command METHOD keyword is used to override this specification for an individual report.
FILESUM	Specifies whether the file summary section of the program listing is to be suppressed or not. The file summary section will be output unless FILESUM NO is specified. FILESUM YES will be ignored if the CONTROL command NOLIST keyword is specified.
INDEF	Specifies whether any in-stream file definition statements are to be listed in the program listing or not. In-stream file definition statements will be listed unless INDEF NO is specified. INDEF YES will be ignored if the CONTROL command NOLIST or NOSOURCE keywords are specified.
INGLOSS	Specifies whether any glossaries for in-stream definitions are to be listed in the program listing or not. Glossaries for in-stream file definitions will be listed unless INGLOSS NO is specified. INGLOSS YES will be ignored if the CONTROL command NOLIST keyword is specified.
INREQ	Specifies whether any input-stream request statements are to be listed in the program listing or not. Input-stream request statements will be listed unless INREQ NO is specified. INREQ YES will be ignored if the CONTROL command NOLIST or NOSOURCE keywords are specified.

CATREQ	Specifies whether any cataloged request statements are to be listed in the program listing or not. Cataloged request statements will be listed unless CATREQ NO is specified. CATREQ YES will be ignored if the CONTROL command NOLIST or NOSOURCE keywords are specified.
MAPREQ	Specifies whether any mapping request statements are to be listed in the program listing or not. Mapping request statements will not be listed unless MAPREQ YES is specified. MAPREQ YES will be ignored if the CONTROL command NOTLIST or NOSOURCE keywords are specified.
SQLSTAT	Specifies whether the SQL statistics section of the program listing is to be suppressed or not. The SQL statistics section will be output unless SQLSTAT NO is specified. SQLSTAT YES will be ignored if the CONTROL command NOTLIST keyword is specified.
MOSAICSTAT	Specifies whether the MOSAIC statistics section of the program listing is to be suppressed or not. The MOSAIC statistics section will be output unless MOSAICSTAT NO is specified. MOSAICSTAT YES will be ignored if the CONTROL command NOTLIST keyword is specified.

Examples

```
LISTCNTL SQLSTAT NO, MOSAICSTAT, NO
```

The SQL statistics section and the MOSAIC statistics section of the program listing will be suppressed.

LISTLIB GLOSSARY Command

```
LISTLIB GLOSSARY {ARRAY | FILE | GROUP | TABLE | VIEW | INVIEW},  
                {ALL | [ITEMS] name ...}
```

The LISTLIB GLOSSARY command is used to list glossaries for definitions contained in the common library. Any given LISTLIB command may only specify one object type, but many LISTLIB commands may be present. This command may only be specified in a definition/dictionary maintenance run.

ARRAY	Specifies that all or selected array definition glossaries are to be listed.
FILE	Specifies that all or selected file definition glossaries are to be listed.
GROUP	Specifies that all or selected transaction group definition glossaries are to be listed.
TABLE	Specifies that all or selected table definition glossaries are to be listed.
VIEW	Specifies that all or selected logical data view glossaries are to be listed.
INVIEW	Specifies that file definition glossaries within all or selected logical data views are to be listed.
ALL	Specifies that glossaries for all objects of the indicated type are to be listed.
ITEMS <i>name ...</i> ITEM	Specifies the names of one or more objects of the indicated type for which glossaries are to be listed.

Examples

```
LISTLIB GLOSSARY ARRAY ALL
```

List the glossaries for all array definitions in the common library.

```
LISTLIB GLOSSARY VIEW ITEMS VIEW1 VIEW2
```

List the glossaries of the logical data views named VIEW1 and VIEW2.

```
LISTLIB GLOSSARY INVIEW ITEM VIEWX
```

List the glossaries of all file definitions contained in logical data view VIEWX.

LISTLIB NAMES Command

LISTLIB NAMES {ALL | ARRAY | FILE | GROUP | TABLE | VIEW}

The LISTLIB NAMES command is used to list the names of definitions contained in the common library. Any given LISTLIB command may only specify one object type, but many LISTLIB commands may be present. This command may only be specified in definition/maintenance run.

ALL	Specifies that the names of all definitions of all types in the common library are to be listed.
ARRAY	Specifies that the names of all array definitions in the common library are to be listed.
FILE	Specifies that the names of all file definitions in the common library are to be listed.
GROUP	Specifies that the names of all transaction group definitions in the common library are to be listed.
TABLE	Specifies that the names of all table definitions in the common library are to be listed.
VIEW	Specifies that the names of all logical data views in the common library are to be listed.

Examples

```
LISTLIB NAMES ALL
```

The names of all items in the common library are listed.

```
LISTLIB NAMES FILE
```

The names of all file definitions in the common library are listed.

MULTILIB Command

MULTILIB {**OFF** | **ORDER** *ddname* ...}

The MULTILIB command is used to disable the search of multiple common libraries when DD statements for multiple libraries are present in the JCL or to specify a specific order in which multiple common libraries are to be searched. Common libraries are identified in the JCL by DD names of M4LIB and M4LIB1 through M4LIB9. If the MULTILIB command is omitted and DD statements for multiple common libraries are included in the JCL, the search order will be M4LIB, and then M4LIB1 through M4LIB9.

OFF	Specifies that the search of multiple common libraries for requested objects is to be disabled. This means that even though the JCL includes DD statements for common libraries, only the common library identified by DD name M4LIB will be searched.
ORDER <i>ddname</i> ...	Specifies the order in which common libraries are to be searched. Acceptable <i>ddname</i> values are M4LIB1 through M4LIB9.

Examples

```
MULTILIB ORDER M4LIB3 M4LIB2 M4LIB1
```

The common libraries associated with DD names M4LIB3, M4LIB2, M4LIB1, and M4LIB are to be searched in that order for any cataloged items.

```
MULTILIB OFF
```

Even if the JCL includes DD statements for multiple common libraries, only the common library associated with DD name M4LIB is to be searched for any cataloged items.

OVERRIDE Command

```
OVERRIDE [DDNAME] old-ddname,  
      WITH new-ddname
```

The OVERRIDE command is used to specify an alternate DD name to be used in place of the standard VISION:Builder DD name for a file. Each overridden DD name must be specified with a different OVERRIDE command. Note that the use of the DDNAME keyword on the various FILE commands performs the same function as the OVERRIDE statement for a given file. The old ddname on the OVERRIDE command is always identical to the default DD name on the FILE command.

DDNAME <i>old-ddname</i>	Specifies the standard VISION:Builder DD name that is to be overridden. DD names M4INPUT and M4LIST may not be overridden with the OVERRIDE command.
WITH <i>new-ddname</i>	Specifies the ddname that is to be used in place of the standard VISION:Builder DD name for the file identified with the DDNAME keyword.

Examples

```
OVERRIDE M4OLD WITH DATA1
```

An override for the DD name of the master file (default DD name of M4OLD) is given. The new DD name for the file is DATA1.

OWNCODE Command

```
OWNCODE [MODULE] module-name,  
        HOOKS hook-number ...
```

The OWNCODE command is used to specify program modules that are to be invoked at certain points within the VISION:Builder processing cycle called hooks. A specified program module may process calls from more than one hook, but any hook number should be specified on only one OWNCODE command.

MODULE <i>module-name</i>	Specifies the name of the program module that will process the hooks specified for the HOOKS keyword.
HOOKS <i>hook-number</i> ... HOOK	Specifies one or more hook numbers that the associated program module is to process. Valid owncode hook numbers are: 10, 11, 20, 21, 30, 50, 51, 60, 61, 62, 63, 70, 91, 92, 93. See the VISION:Builder Environment Guide for the available hook numbers and their functions.

Examples

```
OWNCODE MODULE MYOPS, HOOKS 50 70
```

Program module MYOPS is to be called for own code hooks 50 and 70.

RETRIEVE Command

```
RETRIEVE {ARRAY | FILE | GROUP | REQUEST | TABLE | VIEW | EOF},
        {ALL | [ITEMS] name ...},
        [NEWNAME name]
```

The RETRIEVE command is used to retrieve the source code for objects in a common library. The retrieved source statements are output in the file specified by the DD name M4SSOUT. More than one RETRIEVE command may be provided. A RETRIEVE command is valid only in source statement retrieval runs.

ARRAY	Specifies that the source statements for array definitions are to be retrieved.
FILE	Specifies that the source statements for file definitions are to be retrieved.
GROUP	Specifies that the source statements for transaction group definitions are to be retrieved.
REQUEST	Specifies that the source statements for requests or request groups are to be retrieved.
TABLE	Specifies that the source statements for tables are to be retrieved.
VIEW	Specifies that the source statements for logical data views are to be retrieved.
EOF	Specifies that a /* record is to be output in the M4SSOUT data stream.
ALL	Specifies that the source statements for all items of the specified type that are stored in the common library are to be output.
ITEMS <i>name</i> ... ITEM	Specifies that the source statements for one or more items of the specified type that are stored in the common library are to be output.
NEWNAME <i>name</i>	Specifies the new name that is to be assigned to an item that is retrieved from the common library. When the NEWNAME keyword is present, the ITEMS keyword must also be present and must specify exactly one name.

Examples

```
RETRIEVE GROUP ALL
```

Retrieve the source statements for all transaction group definitions.

```
RETRIEVE FILE ITEM DEFA, NEWNAME DEFX
```

Retrieve the source statements for file definition DEFA, but change the name in the retrieved statements to DEFX.

ROUTE Command

```
ROUTE {[REPORTS] report-name ... | ALL},
      [KEYVALUE 'data-value'],
      TO destination-names ...,
      [DEFER]
```

The ROUTE command specifies the reports, selection data, and destinations to which the reports are to be routed. An application may include many ROUTE commands, and any report may be included in multiple ROUTE commands. A unique ROUTE command is required for each combination of reports, selection data, and destination. Reports that are not identified by any ROUTE command are routed to the default destination. There are no ROUTE command dependencies with regard to the COLLATE command or vice versa. Each of these commands stands alone in their function.

REPORTS <i>report-name</i> ... REPORT	Specifies the reports that are to be routed to the designated destinations. The <i>report-name</i> operand is a list of one or more request names or report names. If the request contains more than one report and only selected reports within the request are to be routed to the designated destinations, the request name can optionally be followed by a colon that is then followed by the report numbers (Rn statement sets) that are to be used. For example, a specification of SALESRPT:135 indicates that only reports 1, 3, and 5 of the request SALESRPT are to be considered for this destination. No embedded blanks are allowed in the list of report numbers. When report names are used, the report number notation does not apply.
ALL	Specifies that all reports in the application are to be routed to the designated destination(s).
KEYVALUE ' <i>data-value</i> ' KEYVAL	Specifies a character constant (a string value enclosed in single quotation marks (')) indicating that only the report data whose routing key value matches the specified data value are to be routed to the designated destination(s). The routing key is a subset of the report key beginning in position 1 of the report key. The length of the KEYVALUE keyword operand specifies the length of the routing key and may include trailing blanks. The length of this value may not exceed the length of the report key.

TO <i>destination-names</i> ...	Specifies one or more destinations to which this report or set of reports is to be routed. Destinations are specified as DD names that correspond to the JCL statements that specify the appropriate data set destination parameters. When the notation 'ddname(member-name)' is used, the destination is assumed to be a PDS and the output is placed in the member identified by member-name. The TO keyword is required with this command. M4LIST may not be used as a report destination.
DEFER	Specifies that the open for the destination data set is to be deferred until just before the first output is to be written to the destination. This would then prevent the creation of empty data sets or unnecessary tape mounts when no data was selected for the report. When this keyword is omitted, the open for the destination data set will occur during program initialization regardless of whether any data was selected for the destination or not. When no data is selected for the report, an empty data set will be created.

Examples

```
ROUTE REPORTS REP1 REP2 KEYVALUE 'A' TO COMMSALES
ROUTE REPORTS REP1 REP2 KEYVALUE 'B' TO GOVTSALES
ROUTE REPORTS REP1 REP2 KEYVALUE 'C' TO COMMSALES
ROUTE REPORTS REP1 REP2 KEYVALUE 'D' TO GOVTSALES
ROUTE REPORTS REP1 REP2 KEYVALUE 'A' TO BRANCH1
ROUTE REPORTS REP1 REP2 KEYVALUE 'B' TO BRANCH2
ROUTE ALL TO ARCHIVE
```

Reports REP1 and REP2 with a keyvalue of 'A' are routed to COMMSALES and BRANCH1; 'B' is routed to GOVTSALES and BRANCH2; 'C' is routed to COMMSALES only, and 'D' to GOVTSALES only. Additionally, all reports are routed to ARCHIVE.

TRACK Command

```
TRACK [NAME] item-name,
      [GENERIC],
      [TYPE {FILE | ARRAY | TABLE | TRAN | REQUEST | REQGROUP}],
      [FILENAME file-name],
      [{EXPIRE mmddy | RETAIN days}],
      [USERID userid]
```

The TRACK command is used to specify that information such as creation date/time, expiration date/time, and other information be maintained for an item. The TRACK command may be used during a processing run in combination with the CATALOG command for maintaining cataloged request and request group information. The TRACK command may also be used during a definition run for maintaining definition information.

NAME <i>item-name</i>	Specifies the name of the item to which this TRACK command applies.
GENERIC	Specifies that the name associated with the NAME keyword is a generic name and not a full name. A generic name must be less than 8 characters long. The use of this keyword implies that the TRACK command parameters apply to all items whose names begin with the generic name.
TYPE <i>item-type</i>	Specifies the type of item designated by the NAME keyword. The item-type is specified by one of the following keywords: <ul style="list-style-type: none"> ■ FILE -- Specifies a file definition item. ■ ARRAY -- Specifies an array definition item. ■ TABLE -- Specifies a table definition item. ■ TRAN -- Specifies a transaction group definition item. ■ REQUEST -- Specifies a request item. ■ REQGROUP -- Specifies a request group item.
FILENAME <i>file-name</i>	Specifies the full or generic file definition name associated with the transaction group name designated by the NAME keyword.

EXPIRE <i>mmddy</i>	Specifies the date on which this item is to expire. Deletion of an item is prevented until the expiration date occurs. The century of the expiration date will be the current century if the expiration date is greater than or equal to the current date. Otherwise, the century of the expiration date will be the next century. For example, if the current date is July 1, 2001 (070101) and the expiration date is specified as 060101, then the real expiration date will be June 1, 2101. The EXPIRE keyword is mutually exclusive with the RETAIN keyword.
RETAIN <i>days</i>	Specifies the number of days the item is to be retained. Deletion of the item is prevented until that number of days has elapsed. The number may not exceed 5 digits.
USERID <i>userid</i>	Specifies the userid of the person responsible for cataloging or updating the item in the common library.

Examples

```
TRACK CUSTDEF, TYPE FILE, EXPIRE 070110
```

Set the expiration date for file definition CUSTDEF to July 1, 2010.

WORK Command

WORK [**AREA**] *number*,
NAME *definition-name*

The WORK command is used to specify the presence of working storage areas allocated for this application. The qualifier W is used to reference data in the working storage areas.

<i>AREA number</i>	Specifies the order in which working storage areas are to be assigned. Working storage areas are assigned contiguous memory space beginning with the lowest area number and upwards. Up to 99 working storage areas may be defined with the area numbers ranging from 1 to 99.
<i>NAME definition-name</i>	Specifies the name of the file definition that defines the detail fields within a working storage area. When multiple working storage areas are defined, the field names within all working storage definitions must be unique.

Examples

```
WORK 1 NAME WORKDEF1
```

Allocate a work area assigned as number 1 that is defined by file definition WORKDEF1.

Procedural Command Group

This chapter describes each command and available built-in functions in detail. After the syntax and operands of a command are explained, several examples of the command are given. Values for fields are given, as well as other pertinent information and the result of the procedure statements.

The following list summarizes the ASL commands for procedure statements. Longer command and keyword names have abbreviations that are defined below the command or keyword name.

Command	Function
CALL	Calls another procedure or invokes report or subfile output.
CASE	Begins a group of procedure statements within a DO CASE block that are to be performed when the CASE condition is true.
COMBINE COM	Concatenates two or more character strings into a 1-character field.
CONTINUE CONT	Continues by going to the end of the procedure.
DO	Either initiates a conditional loop within a procedure or begins a block of procedure statements containing groups of CASE commands. Only the first CASE block within the DO CASE block with a true condition is executed.
ELSE	Begins a group of statements within an IF or DO CASE block that are to be performed when all conditions are false.
END	Terminates a DO, IF, or PROC block.
FIELD FLD	Defines a temporary field within a procedure.
GO	Jumps to a specified statement within a procedure.
IF	Begins a block of procedure statements that are performed when the condition on the IF command is true.
INCLUDE	Used to include a cataloged request into the application.

Command	Function
LEAVE	Provides a way to exit a DO loop immediately.
LET	Sets the value of a field equal to the value of a field, constant, or arithmetic expression.
LOCATE LOC	Locates a cell, row, or column of an array. Note: LOCATE is available in <i>VISION:Builder</i> and <i>VISION:Two</i> , but not <i>VISION:Inform</i> .
PROC	Begins a new procedure block.
RELEASE REL	Releases a segment or an array occurrence.
REPLACE REP	Replaces defined characters within a character field with other defined characters.
RETURN RET	Returns control to the calling procedure.
TRANSFER	Causes control to be transferred to a specific point within the processing cycle.

Quotation Marks

When field values are shown in coding examples, the contents of the field are enclosed in single quotation marks. For example, the contents of the field CUSTNO are enclosed in single quotation marks, but the quotation marks are not part of the field.

```
Field:          CUSTNO
Contents:      '00001'
```

The quotation marks delimit the beginning and end of the field.

For the rules and explanations of notational conventions and page layout, see [Chapter 2, Terminology, Syntax, and Processing](#).

Built-In Functions

This section describes the built-in functions that are available with the procedural commands. After the syntax and operands of a function are explained, several examples of the function are given. Values for the fields are given, as well as other pertinent information and the result of the procedure statements.

A function is a subprocedure or subprocess that derives a value or condition from other data. The function value is created when reference is made to it. You use the function as if it is a field. You write a function differently than a field, but basically you use a function wherever you use a field name.

Value Functions and Conditional Functions

Functions return either “value” (that is, they represent a field value) or “conditional” (that is, they represent a true or false condition). When needed, you use functions in procedure statements.

- You can use value functions anywhere a field name can be used.
- You can use conditional functions anywhere a conditional operand is needed.

Specifying Functions

There are five built-in functions available in ASL. You specify functions by entering a function name followed immediately (with no intervening spaces) by a left parenthesis, one or more keyword phrases, and terminating with a right parenthesis.

Longer function and keyword names have abbreviations that are defined directly beneath the function or keyword in the following descriptions.

Types of Built-In Functions

There are two types of built-in functions: conditional and value.

Conditional Functions

All conditional functions are evaluated and return a true or false condition, but some functions also perform other actions. The conditional functions are as follows:

FIND	Find a segment function. This function returns a true condition when a segment or record exists. If the segment or record does not exist, the function returns a false condition. If the segment or record exists, the function locates the segment or reads the record so that subsequent field values can be obtained from the segment.
-------------	---

LOCATE	Locate a cell, row, or column in an array. This function returns a true condition when the row and/or column specifications are within the bounds of the array. Otherwise, the function returns a false condition. If the function returns a true condition, the specified cell, row, or column is available for subsequent processing. Note: LOCATE is available in <i>VISION:Builder</i> and <i>VISION:Two</i> , but not <i>VISION:Inform</i> .
SCAN	String scan function. This function searches a field for a character or set of characters. <ul style="list-style-type: none">■ If the characters being searched for are found, the function returns a true condition.■ If the characters are not found, the function returns a false condition. The location where the characters are found is kept in flag fields when the function is true. When the function is false, the flag fields are not set.
VALIDATE VAL	Field validation function. This function validates the contents of a field for either a valid calendar date or for a predefined pattern of characters. <ul style="list-style-type: none">■ If the date is valid or the pattern is found, the function returns a true condition.■ If the content is not a valid date or the pattern is not satisfied, the function returns a false condition.

Value Functions

Value functions return an actual value, either a result value from a table or a part of an existing field. The value functions are as follows:

LOOKUP LU	Table lookup function. This function processes external tables by searching an argument list and returning back a result value.
PF	Partial field function. This function isolates part of a character field.

FIND Function (Conditional)

```
FIND ( [SEGMENT] segment-name
      [FIRST|LAST|NEXT|WHERE selection-expression])
```

Use FIND as a conditional function in a logical expression to establish the existence of an occurrence of a segment or record.

- If an occurrence of the segment or a record exists, the FIND function returns a true condition and that segment or record is used for processing in the current procedure.
- If an occurrence of the segment or a record cannot be found, the FIND returns a false condition and reference should not be made to fields on the missing segment or record.

There are five keywords:

SEGMENT <i>segment-name</i>	Specifies the name of the segment to locate.
SEG	<ul style="list-style-type: none"> ■ If the segment-name is a root segment, a record is read from the file. ■ If the segment-name is a lower level segment, a particular occurrence of the segment is located. <p>If the WHERE phrase is not present, the following rules apply:</p> <ul style="list-style-type: none"> ■ The first FIND function for a segment in the procedure locates the first occurrence of the segment. ■ Subsequent FIND functions for the same segment in the same procedure locate segments in the order that they are in the record. A subsequent FIND function for a record reads the next record on the file. When the subsequent FIND function is processed, the segment previously found is no longer available. In other words, only one occurrence is processed at any given time. ■ For some databases (for example, DB2®), the concept of “first” and “order” might have no meaning. In these cases, the FIND function will locate an occurrence of a segment/record. Subsequent FIND functions will locate a different occurrence of the segment/record.

FIRST	Causes the first occurrence of a lower level segment to be located for processing. This keyword is not permitted for memory optimized (MOSAIC) processing of relational or Generalized Data Base Interface (GDBI) files.
LAST	Causes the last occurrence of a lower level segment to be located for processing. This keyword is not permitted for memory optimized (MOSAIC) processing of relational or GDBI files.
NEXT	Causes the next available occurrence of a lower level segment to be located for processing. The first FIND NEXT function of a procedure locates the first occurrence of a segment.
WHERE <i>selection-name</i>	<p>Specifies the value of the segment or record key to locate. Use a logical expression as the WHERE operand. Use a primary key field name for the segment or record as one of the field names in the logical expression.</p> <p>For a lower level segment, the WHERE phrase causes only one segment to be located: the segment whose primary key is equal to the value specified in the operand.</p> <p>For a root segment, the WHERE phrase causes only one record to be read and is only allowed for a direct-read file. You can use two types of reads: the key of the record equal to the value specified in the operand, or the key of the record whose value is greater than or equal to the value specified in the operand.</p>

Example

The FIND function is a conditional function you can use anywhere in a logical expression. The examples show the FIND function on an IF command. For more information on the IF command, see [Chapter 4, Procedural Command Group](#).

```
IF FIND(1.SEGX)
  LET T.TEMP1 = 1.FIELD1
ELSE
  LET T.TEMP1 = 0
END
```

The FIND function reads a record from file 1. If such an occurrence is found, the FIND function is true and the next statement is executed; if no occurrence is found, the FIND function is false and the ELSE clause is executed.

```
IF FIND(SEGMENT ORDER WHERE ORDERNUM = '12345')
  CALL PROC CHKORD
END
```

The FIND function searches for an ORDER segment with the key value of 12345. ORDER is a lower level segment in the file.

- If the function finds a segment, the PROC CHKORD is called.
- If the function does not locate the segment, processing continues with the statement after the END command.

LOCATE Function (Conditional)

Note: LOCATE is available in *VISION:Builder* and *VISION:Two*, but not *VISION:Inform*.

```
LOCATE( [ARRAY] array-identifier  
        {[Row row-number] [COLUMN column-number]})
```

Use LOCATE as a conditional function in a logical expression to locate a cell, row, or column in an array. It operates exactly like the LOCATE command, except that the function returns a true or false condition. You use this condition in evaluating the logical expression.

Use the function in place of the command where the row and/or column parameters might cause the LOCATE command to fail. Use the LOCATE function to perform an array location operation and test for its success or failure in one operation.

For a discussion of the LOCATE function keywords, see the section [LOCATE Command](#).

Examples

```
IF LOCATE (ARRAY A ROW T.Y COLUMN T.X)  
  ; process cell  
ELSE  
  ; Handle error situation - check ASTATUS flag field  
END
```

LOOKUP Function (Value)

```
LOOKUP ( [TABLE] table-name [ARGUMENT] lookup-argument  
[NEAREST | SMALLER | LARGER | INTERPOLATE])
```

Note: If you do not choose any of the following keywords, the arguments in the table are searched for a value equal to the value in lookup-argument.

LOOKUP LU	Is a value function that takes the given argument value, searches the argument list in the table to locate a specific argument, and returns the corresponding result value. Because LOOKUP is a value function, you can use it anywhere you use a field name or constant. There are six keywords.
TABLE <i>table-name</i> TAB	Specifies the name of a table. Specify the operand to be the name of a table definition defined to the VISION:Inform library.
ARGUMENT <i>lookup-argument</i> ARG	Specifies the field containing the argument value. Use a field name as the operand.
NEAREST NRST	Causes the arguments in the binary table to be searched for a value equal to or nearest the value in lookup-argument. This keyword has no operands.
SMALLER SMLR	Causes the arguments in the binary table to be searched for a value equal to or smaller than the value in lookup-argument. This keyword has no operands.
LARGER LRGE	Causes the arguments in the binary table to be searched for a value equal to or larger than the value in lookup-argument. This keyword has no operands.
INTERPOLATE INT	Performs linear interpolation on the argument list in the binary search table to determine the argument / result to be selected. This keyword has no operands.

As the LOOKUP function represents a value, you can use the function anywhere you would use a field name or a constant. In the examples, the LOOKUP function is used on a LET command. For more information on the LET command, see [Chapter 4, Procedural Command Group](#).

Example 1

Table Name: MONTHS

Argument list	Result list
01	JANUARY
02	FEBRUARY
03	MARCH
04	APRIL
05	MAY
06	JUNE
07	JULY
08	AUGUST
09	SEPTEMBER
10	OCTOBER
11	NOVEMBER
12	DECEMBER

MM is a field in a file whose contents is 11.

```
LET T.MONTH = LOOKUP (MONTHS, ARGUMENT MM)
```

After the LOOKUP function, T.MONTH contains NOVEMBER.

Example 2

Table Name: GRADES

Argument list	Result list
1.0	FAILING
1.5	BELOW AVERAGE, NEEDS IMPROVEMENT
2.0	BELOW AVERAGE
2.5	AVERAGE
3.0	GOOD
3.5	EXCELLENT
4.0	PERFECT

SCORE is a field in the file whose content is 3.8.

```
LET T.EVALUATE = LOOKUP (GRADES ARGUMENT SCORE NRST)
```

After the LOOKUP function, T.EVALUATE contains PERFECT.

PF Function (Value)

PF ([FIELD] *field-name* [START] *start-position* [LENGTH *partial-length*])

PF	Is a value function that defines the value as a part of the character field against which the function is used. You can use the PF function on a character field and in any replacement or logical expression. There are three keywords:
FIELD <i>field-name</i> FLD	Specifies the name of a character field (fixed or variable length). The PF function returns a value that is only part of the field specified in this operand.
START <i>start-position</i>	Is a required entry that specifies the starting character position of the value within the previously named field, where byte 1 is the beginning of the field. This operand can be an integer or one of the flag field names LS, LN, MS, MN, RS, or RN, which are explained in the section SCAN Function (Conditional) .
LENGTH <i>partial-length</i> LEN	Is required for a fixed length field, but is an optional entry for a variable length field. The result of the PF function is a partial value from the field named in the first operand. The partial-length specifies how many characters the partial value is to contain. This operand can be an integer or one of the flag field names LS, LN, MS, MN, RS, or RN, which are explained in the section SCAN Function (Conditional) . You can omit the keyword LENGTH and its operand for variable length fields, in which case, the remaining length of the field from the start position is assumed.

Example 1

Assume these values for the following fields:

```
CHARS      = 'ABCDEFGHIJKL'
T.NUMBER   = '3'
LS         = '2'
LN         = '10'
```

Function	Partial Field Value
PF(CHARS START 1 LENGTH 7)	'ABCDEFG'
PF(CHARS 3 6)	'CDEFGH'
PF(CHARS LS 4)	'BCDE'

The PF function on the CHARS field causes the value of the field to be a subset of the original contents of the field.

Example 2

Field: CUSTNO PF(CUSTNO START 1 LENGTH 1)
Field contents: '1000' '1'

```
IF PF(CUSTNO START 1 LENGTH 1) = '1' THEN  
    CALL REPORT CUSTOMER  
END
```

The PF function on CUSTNO causes the first position to be isolated for testing on the IF command.

This example shows the partial field function being used where a field name or constant could be used as a part of a logical expression. The PF function represents a value to be tested against another value. In the case shown, the condition will be true.

Example 3

The field VARIABLE is a variable length field.

Field: VARIABLE PF(VARIABLE,3) PF(VARIABLE,2,4)
Field contents: 'ABCDEFGHIJ' 'CDEFGHIJ' 'BCDE'

The value of the partial field is the part of field VARIABLE defined by a starting position and a length parameter. For variable length fields, you can omit the length parameter to signify the remainder of the field.

SCAN Function (Conditional)

```
SCAN( [FIELD] field-name [FOR] search-value/pattern
      {FROM LEFT | FROM RIGHT} [NOTEQUAL])
```

SCAN is a conditional function. This function checks a field to see if a specified field contains a specific character string or pattern of characters. If a match is found, a true condition for the function is established. If no match is found, the condition is false.

When the function finds the character string or pattern, the location of the search-value/pattern within the field being scanned is stored in the following flag fields:

Field Flag Name	Long Form Field Flag Name	Description
LS	LSTART	Left part Start
LN	LNUMBER	Left part Number of characters
MS	MSTART	Middle part Start
MN	MNUMBER	Middle part Number of characters
RS	RSTART	Right part Start
RN	RNUMBER	Right part Number of characters

The scanned field can be considered as having the following parts:

- Search-value/pattern.
- Part of the field to the left of the search-value/pattern.
- Part of the field to the right of the search-value/pattern.

The flag fields identify the starting location and length of these three parts of the field being scanned.

- LS contains the starting location of the part of the field being scanned that is to the left of the search-value/pattern.
- LN contains the length of this left portion of the field.
- MS contains the location of the beginning of the search-value/pattern within the field being scanned.
- MN contains the length of the search-value/pattern.
- RS contains the starting location of the part of the field being scanned that is to the right of the search-value/pattern.
- RN contains the length of this right portion of the field.

There are four keywords:

FIELD <i>field-name</i> FLD	Specifies the field to scan. Specify a name or PF function as the operand.
FOR <i>search-value/pattern</i>	Specifies the value or pattern for which to scan. Specify a name, character constant, or validation pattern as the operand. For valid pattern symbols, see Appendix C, Technical Notes .
<u>FROM LEFT</u> FROM RIGHT	Specifies the direction of the scan. If you omit this keyword phrase, LEFT is assumed.
NOTEQUAL NE	If you add the NOTEQUAL keyword to the SCAN function, the function returns a true condition when the search-value/pattern is not found. If you do not enter the NOTEQUAL keyword, the SCAN function returns a true condition when the search-value/pattern is found. This keyword has no operands.

Example 1

Assume these values for the following fields:

	ALPHA	=	'ABCCAB123ABEEE'		
Initial					
Values:	LS	=	'0'	LN	= '0'
	MS	=	'0'	MN	= '0'
	RS	=	'0'	RN	= '0'

Function	RESULTS						
	Condition	LS	LN	MS	MN	RS	RN
SCAN(ALPHA FOR 'AB' FROM LEFT)	TRUE	01	00	01	02	03	13
SCAN(ALPHA FOR 'AB' FROM RIGHT)	TRUE	01	10	11	02	13	03
SCAN(ALPHA FOR 'DE' FROM LEFT)	FALSE	01	15	16	00	16	00
SCAN(ALPHA FOR P'999' FROM LEFT)	TRUE	01	07	08	03	11	05
SCAN(ALPHA FOR P'999' NE)	FALSE	00	15	16	00	00	00

The first occurrence of the search value/pattern located in the field causes the SCAN function to be true, and the flag fields are set to that location.

The next two examples of the SCAN function show it being used in the IF command where a logical expression is necessary. You can use the SCAN function in a logical expression, because it sets a true or false condition. For more information on the IF command, see [Chapter 4, Procedural Command Group](#).

Example 2

```
IF SCAN(T.MSG FOR 'XYZ')
  LET T.LEFTPART = PF(T.MSG LS LN)
END
```

If the field T.MSG contains THE XYZ COMPANY, the SCAN function is evaluated as true, because the letters XYZ were found in T.MSG. As a result of the expression, T.LEFTPART contains the value THE b/. Because the FROM phrase is omitted, FROM LEFT is assumed.

Example 3

```
IF SCAN(CUSTNO FOR P'99999')
  CALL REPORT VALID
ELSE
  CALL REPORT INVALID
END
```

The field CUSTNO is a 5-byte character field. The pattern P'99999' checks for numeric characters. If the field CUSTNO contains any non-numeric characters, the SCAN function is evaluated as false.

VALIDATE Function (Conditional)

VALIDATE([FIELD] *field-name* {PATTERN P'*pattern*' | DATE})

VALIDATE
VAL

Specify conditional function. It validates a field for a pattern or date.

- If the field contains a valid date or its content matches the specified validation pattern, the function is true.
- If the field does not contain a valid date or the specified validation pattern, the function is false.

Use the VALIDATE function as a logical expression in procedure statements. There are three keywords:

FIELD *field-name*
FLD

Specifies the field to be validated. Use the name of a field or a PF function of a field as the operand.

PATTERN P'*pattern*'
PAT

Specifies a pattern to use for the validation.

- Specify a pattern starting with the letter P, followed by a beginning single quotation mark, a string of special pattern symbols, and a closing single quotation mark.
- Use up to 30 characters in the operand.
- Code one pattern symbol for each character of the field being validated.

For valid pattern symbols, see [Appendix C, Technical Notes](#).

The function checks each character in the specified field against its corresponding pattern character. This keyword is mutually exclusive with the DATE keyword.

DATE

Checks that the field contains a valid calendar date. This keyword causes the function to validate the content of the field for a valid calendar date.

- If the field contains a valid date, the function is evaluated as true.
- If the field contains a non-calendar date or the date is not in the format expected by the function (as defined to the system), the function is evaluated as false.

The type of date storage (for example, Julian and MMDDYY) is defined in M4SFPARM. This keyword has no operands and is mutually exclusive with the PATTERN keyword.

Example 1

Function	If ORDRDATE Contains	Condition
VALIDATE(ORDRDATE PAT P'999999')	'123456'	TRUE
VALIDATE(ORDRDATE DATE)	'987654'	FALSE
VALIDATE(ORDRDATE PAT P'999999')	'011595'	TRUE
VALIDATE(ORDRDATE DATE)	'011595'	TRUE
VALIDATE(ORDRDATE PAT P'999999')	'ABCDEF'	FALSE
VALIDATE(ORDRDATE DATE)	'ABCDEF'	FALSE

The next two examples show the conditional VALIDATE function in the IF statement, which requires a logical expression. See [Chapter 4, Procedural Command Group](#) for more information on the IF command.

Example 2

```
IF VALIDATE (FIELD ORDRDATE DATE)
  CALL REPORT ORDER
ELSE
  LET T.MSG = 'ILLEGAL DATE'
END
```

If the field ORDRDATE contains 999999, the VALIDATE function would be evaluated as false, because 999999 does not represent a valid calendar date.

Example 3

```
IF VALIDATE (PF (NUM 5 4) P'9999')
  LET T.COUNT = PF (NUM 5 4)
END
```

This section describes each procedural command in detail.

CALL Command

```
CALL { [PROCEDURE] procedure-name |  
      REPORT report-name |  
      SUBFILE subfile-name |  
      MODULE 'module-name' |  
      CEEDATE | CEEDATM | CEEDAYS | CEEDYWK | CEEGMT |  
      CEEGMTO | CEEISEC | CEELOCT | CEEQCEN | CEESCEN |  
      CEESECI | CEESECS | CEEUTC  
      [USING parm ...] }
```

Use the CALL statement to branch to subprocesses such as other internal procedures, external user written modules, reports, and subfiles. When the subprocess, invoked by the CALL command completes, the subprocess passes control back to the procedure statement following the CALL command.

The keywords beginning with “CEE” designate selected IBM® Language Environment® (LE) services that the CALL command specifically recognizes and assists with parameter conversion where necessary. Specifically, the CALL command automatically converts any parameter expected to be a length-prefixed character string (VSTRING) for any of these CEE... services whenever the CEE... keywords are used.

Note: The IBM Language Environment (LE) services, CEE..., are Year 2000 compliant.

See the IBM Language Environment for MVS & VM Programming Reference manual for syntax and examples of these callable services.

LE services can also be invoked using the “MODULE” keyword. However, no parameter conversion takes place when this form is used.

PROCEDURE <i>procedure-name</i> PROC	Specifies a procedure to be executed. Use the name of a procedure within the current application as the operand.
REPORT <i>report-name</i> REP	Specifies that a report is to be output. Use the name of a report in the current application as the operand.
SUBFILE <i>subfile-name</i> SUB	Specifies that data is to be output to a subfile. Use the name of a subfile in the current application as the operand.

MODULE <i>module-name</i> MOD	Specifies an external user written load module to call. Use an external load module name as the operand (module-name).
CEEDATE	LE service that converts dates in the Lilian format to character values.
CEEDATM	LE service that converts number of seconds to character timestamp.
CEEDAYS	LE service that converts character date values to the Lilian format. Day one is 15 October 1582, and the value is incremented by one for each subsequent day.
CEEDYWK	LE service that provides day of week calculation.
CEEGMT	LE service that gets current Greenwich Mean Time (date and time).
CEEGMTO	LE service that gets difference between Greenwich Mean Time and local time.
CEEISEC	LE service that converts binary year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 15 October 1582.
CEELOCT	LE service that retrieves current date and time.
CEEQCEN	LE service that queries the century window.
CEESCEN	LE service that sets the century window.
CEESECI	LE service that converts a number representing the number of seconds since 00:00:00 15 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond.
CEESECS	LE service that converts character timestamps (a date and time) to the number of seconds since 00:00:00 15 October 1582.
CEEUTC	LE service that performs the same function as CEEGMT.

USING *parm...*

Specifies any parameters to be passed to the MODULE or CEE... service that was named in the previous keyword phrase.

Use field names or constants as the operand. Except as noted above, use parameter data types that conform to the data types expected by the called program.

Example 1.

```
CALL PROCEDURE AVGSAL
```

This CALL command calls a procedure named AVGSAL, which is defined as a subroutine procedure. After AVGSAL executes, it passes control to the statement after the CALL to the subroutine procedure.

Example 2.

```
CALL REPORT NAMELIST
```

This CALL command calls for output to a report named NAMELIST before continuing within the procedure.

Example 3.

```
CALL SUBFILE MFDATA
```

This CALL command calls for a record to be output to the subfile MFDATA before continuing in the procedure.

Example 4.

```
CALL MODULE MYPROG USING AMOUNT FIELDX 'ABC'
```

This CALL command calls for an external module of code named MYPROG to be executed. The following parameters will be passed.

AMOUNT contains: '10'

FIELDX contains: '011501'

A literal value: 'ABC'

The first parameter is the contents of field AMOUNT, which is 10; the second parameter is the contents of FIELDX, which is 011598; and the third parameter is the literal constant ABC.

Example 5.

```
CALL CEEDAYS USING BRTHDATE 'MM/DD/YY' T.LILDATE T.FEEDBACK
```

This CALL command converts the date in the field BRTHDATE to a Lilian date. The string 'MM/DD/YY' is a picture string representing the format of the contents of the BRTHDATE field.

Example 6.

```
CALL CEEDATE USING T.LILDATE 'DD Mmm YYYY' T.NEWDATE T.FEEDBACK
```

This CALL command converts the Lilian date in temporary field LILDATE into character format in the temporary field NEWDATE. The picture yields a date such as 15 Jan 2001.

CASE Command

CASE [**WHERE**] *logical-expression*

The CASE command begins a group of procedure statements within a DO CASE block. The procedure statements are performed when the condition of the CASE command is true.

The program performs only the first group of procedure statements whose CASE command condition is true within the DO CASE block. Once the group of procedure statements executes, control is transferred immediately to the END command corresponding to the DO CASE block.

WHERE *logical-expression* Specifies the logical expression to be evaluated to determine whether the following group of statements is to be performed.

If the logical expression is true, control passes to the statement immediately following the CASE command; otherwise, control passes to the next CASE, ELSE, or END command within the current DO CASE block.

Example.

```
LET T.TEMP1 = 0
DO CASE
  CASE WHERE CUSTNO EQ '00001'
    LET T.TEMP1 = 1
  CASE WHERE CUSTNO GT '00001'
    LET T.TEMP1 = 2
  ELSE
    LET T.TEMP 1 = -1
END
```

This DO CASE block sets the value of T.TEMP1 to:

- 1 if CUSTNO is 00001.
- 2 if CUSTNO is greater than 00001.
- 1 for all other cases (that is, CUSTNO is less than 00001).

COMBINE Command

```
COMBINE [FIELDS] field1 ... STORE result-field [[BLANKS] number]
```

COMBINE COM	The COMBINE command concatenates two or more character strings and store the result in a specified destination field. You can also specify the number of blanks inserted between each string.
FIELDS <i>field1</i> FLDS	Specifies a list of character string fields. Make operands the names of character string fields (fixed or variable length) or literals enclosed in quotation marks as operands.
STORE <i>result-field</i>	Specifies the result field. Make the operand the name of a character string field (fixed or variable length) to contain the concatenation of all the literals and fields named in the FIELDS keyword phrase.
BLANKS <i>number</i>	Specifies the number of blanks to insert between each pair of strings. Use an integer constant as an operand. If you omit this keyword phrase, the default is zero (no blanks between fields).

Example 1

```
COMBINE '(' AREA ')' PHONE STORE RESULT
```

For the following field values, the COMBINE concatenates the literal '(' to the field AREA with no spaces in between:

Fields	AREA	PHONE	RESULT
Contents before:	'213'	'555-1212 '	'9999999999999'
Contents after:	'213'	'555-1212'	'(213)555-1212'

Then, the literal ')' is concatenated, followed by the contents of the PHONE field. The result of the concatenations is stored in field RESULT.

Example 2

```
COMBINE FIELDS PF(AREA 1 1) PF(AREA 2 1) PF(AREA 3 1) ,
STORE T.AREA BLANKS 1
```

For the following field values, the concatenation is done with one blank between each field:

Fields	AREA	T.AREA
Contents before:	'213'	'99999'
Contents after:	'213'	'2 1 3'

The partial field function is used on the fields in the FIELDS keyword phrase. The command is also coded on two lines using a comma at the end of the first line to denote a continuation of the procedure statement.

CONTINUE Command

CONTINUE
CONT

The CONTINUE command causes the normal record processing cycle to continue with the next occurrence of the segment within the controlling set of segments. (See [Chapter 2, Terminology, Syntax, and Processing](#), section [Implicit Loops and Set Operation](#).)

Note that this action is equivalent to a RETURN command if this procedure has been initiated by another procedure and there are no more occurrences of the segment.

Example 1

```
IF CUSTNO EQ '00001'  
    LET T.COUNT = T.COUNT + 1  
    CALL REPORT LISTX  
ELSE  
    CONTINUE  
END  
CALL SUBFILE SUBF1
```

A CUSTNO value of 00001 causes the LET command to execute followed by the CALL command to the LISTX report. Then, the CALL command to the subfile SUBF1 is executed.

Any other value for CUSTNO causes the CONTINUE command to execute, and the CALL command to the subfile SUBF1 is not executed.

Example 2

```
IF CUSTNO LT '10000'  
    CALL REPORT FIRST  
    CONTINUE  
END  
CALL REPORT SECOND
```

A value of CUSTNO less than 10000 causes the CALL command to report FIRST to execute. The CONTINUE bypasses the CALL command to report SECOND. A value of CUSTNO greater than or equal to 10000 causes the CALL command to report SECOND to execute.

DO Command

```
DO { [WHILE logical-expression]
    [UNTIL logical-expression]
    [FORALL segment-name]
    { [FORALL CELLS IN ARRAY array-identifier]
      [FORALL COLUMNS IN ARRAY array-identifier
        [WITHIN ROW row-number]]
      [FORALL ROWS IN ARRAY array-identifier
        [WITHIN COLUMN column-number]] }
    [FOR integer] } |
    [CASE]
```

Array keywords are available in VISION:Builder and VISION:Two, but not VISION:Inform.

The DO command begins a group of procedure statements called a DO block. The procedure statements within the DO block define an explicit procedural looping structure, the exception being individual CASE blocks.

Note that implicit looping, as discussed in [Chapter 2, Terminology, Syntax, and Processing](#), still occurs within DO CASE blocks.

The keyword phrases that follow the DO command determine which type of processing is performed. The DO block must have a corresponding END command.

WHILE logical-expression Specifies a logical expression that is evaluated at the beginning of each iteration of the explicit loop (that is, at the DO command).

- If the condition is true, the program performs another iteration of the explicit loop performed.
- Otherwise, the program terminates the explicit loop and transfers control to the statement after the END command associated with the DO block.

Change the conditions in the logical expression within the DO block in order to terminate the explicit loop.

Do not use the WHILE keyword phrase with the CASE keyword.

<p>UNTIL <i>logical-expression</i></p>	<p>Specifies a logical expression that is evaluated at the end of each iteration of the explicit loop (that is, at the END command).</p> <ul style="list-style-type: none">■ If the logical expression is true, the program terminates the explicit loop and transfers control to the statement after the END command associated with the DO block.■ Otherwise, the program performs another iteration of the explicit loop. <p>Change the conditions in the logical expression within the DO block in order to terminate the explicit loop.</p> <p>Do not use the UNTIL keyword phrase with the CASE keyword.</p>
<p>FORALL <i>segment-name</i></p>	<p>Specifies the name of a segment.</p> <ul style="list-style-type: none">■ If the segment is a root segment from a user-read additional file, the program reads a record from the file at each iteration of the DO block. The processing of the DO block finishes at the end of all occurrences of the segment for the current record.■ If the segment is a lower level segment, the program executes the DO block for all of the segment occurrences. <p>Each iteration of the DO block processes the next occurrence of the specified segment.</p> <p>Do not use the FORALL keyword phrase with the CASE keyword.</p>
<p>ARRAY <i>array-identifier</i> ARR</p>	<p>Specifies the array-identifier for the FORALL keyword. Use the name of an array in the current application as the operand.</p>
<p>ROW <i>row-number</i></p>	<p>Specifies the row of the array for the FORALL keyword. Use a constant, field name, or arithmetic expression, whose value specifies the row number to locate, as the operand.</p>
<p>COLUMN <i>column-number</i> COL</p>	<p>Specifies the column of the array for the FORALL keyword. Use a constant, field name, or arithmetic expression, whose value specifies the column number to locate, as the operand.</p>

FOR *integer*

Specifies the maximum number of times the program executes the DO block. Use an integer value from 1 to 99999.

If the conditions set up under the WHILE or UNTIL operands have not been met within the specified number of interactions on the FOR keyword phrase, the program automatically terminates the DO block at the maximum number.

Do not use the FOR keyword phrase with the CASE keyword.

CASE

Specifies that CASE command blocks follow. This keyword has no operands. You can have an ELSE command in a DO CASE block.

- The program executes the first CASE command whose conditions are met within the DO CASE block and passes control to the corresponding END command.
- The CASE operand does not cause any explicit loop processing like the WHILE, UNTIL, and FORALL keyword phrases. However, all implicit looping, as discussed in [Chapter 2, Terminology, Syntax, and Processing](#), still occurs.

Do not use the CASE keyword with the WHILE, UNTIL, FORALL, or FOR keywords.

If the condition in the WHILE operand is initially false, the program does not execute the DO block. Conversely, the UNTIL keyword makes it possible for the program to execute the statements within the DO block at least once even though the condition is initially true.

You can use the WHILE, UNTIL, FORALL, and FOR keywords in combination with each other.

If you specify both the FORALL and WHILE keywords, the program retrieves the first occurrence of the segment before the WHILE condition is evaluated.

Example 1

```
LET T.COUNT = 0
DO WHILE T.COUNT = 0
  CALL REPORT SAMPLE
  LET T.COUNT = 1
END
```

In the example above:

- The program sets the field T.COUNT to zero before starting the DO block; hence the condition in the DO command is true and the program executes the procedure statements within the DO block.
- In the second interaction of the explicit loop, the condition in the DO command fails and the program bypasses the procedure statements.
- The program passes control to the next procedure statement after the END for the DO block.
- The program processes the CALL command within the DO block only once each time the procedure is executed.

Example 2

```
LET T.RECNUM = 0
DO UNTIL T.RECNUM EQ 3
  CALL SUBFILE RECOUT
  LET T.RECNUM = T.RECNUM + 1
END
```

Because the program sets T.RECNUM to zero before starting the DO block, the program executes the procedure statements within the DO block.

- At the end of the DO block, T.RECNUM contains a 1 and the UNTIL condition fails, allowing the DO block to be executed a second time.
- After the DO block executes a second time, T.RECNUM contains a 2 and the UNTIL condition fails again, allowing the DO block to be executed a third time.
- After the DO block executes a third time, T.RECNUM contains a 3 and the UNTIL condition is true, causing the DO UNTIL to be satisfied.

Processing continues with the procedure statement after the END command corresponding to the current DO command.

Example 3

```
LET T.FIELD1 = 0
DO FORALL 1.ITEM
  LET T.FIELD1 = 1.FIELD1 + T.FIELD1
END
```

The segment 1.ITEM is a root segment from a user-read additional file. The program reads a record from the file at the beginning of the DO block.

- If there is a record in the file, the program executes the procedure statements within the DO block.
- If there are no records to read, the DO block fails and the program bypasses the procedure statements in the DO block. The program reads another record for each execution of the loop and the previous record is no longer available to process.

The program executes the DO block once for each record in file 1. Therefore, the field T.FIELD1 accumulates a total of all of the values of 1.FIELD1 from each record of file 1. The program terminates the DO block at the end of file on file 1 and then executes the statement following the END command.

For all files read with the DO FORALL, the program reads the file until EOF is reached. The last record read is not kept for processing after the DO block.

Example 4

```
LET T.AMOUNT = 0
DO FORALL ROWS IN ARRAY A.XYZ WITHIN COLUMN T.COLNO,
    FOR 50,
    WHILE A.CLASS = 'X',
    UNTIL T.AMOUNT > 5000
    LET T.AMOUNT = T.AMOUNT + A.AMOUNT
END
```

This example loops through up to 50 rows of column T.COLNO as long as A.CLASS contains an X and until T.AMOUNT exceeds 5000.

Example 5

```
DO CASE
CASE CLASS = 1
    LET RATE = RATE * 1.05
CASE CLASS = 2
    LET RATE = RATE * 1.10
END
```

For the following field values, the program evaluates the first CASE command and the logical expression is false:

Field:	CLASS	RATE
Contents before:	'2'	'5.00'
Contents after:	'2'	'5.50'

The program evaluates the second CASE command and the logical expression is true. The program executes the procedure statement following the second CASE command and passes control to the END command corresponding to the current DO CASE block.

The program executes the procedure statements following the first CASE command evaluated with a true logical expression and immediately passes control to the END command for the DO CASE block.

For the same code with the following field values, all of the logical expressions in the CASE commands are evaluated as false:

Field:	CLASS	RATE
Contents before:	'0'	'1'
Contents after:	'0'	'1'

Since an ELSE command is not coded, the program immediately passes control to the END command for the DO block.

ELSE Command

ELSE

The ELSE command begins a group of procedure statements within an IF or DO CASE block that execute if the conditions in the preceding IF command, or all of the preceding CASE commands, are false. The ELSE command within a DO CASE block must follow all of the CASE commands within the same DO CASE block. There are no keywords.

Example 1

```
IF PF (CUSTPH 1 3) = '713'
  LET T.TITLE = 'HOUSTON'
ELSE
  LET T.TITLE = 'OUTSIDE OF HOUSTON'
END
```

For the following field values, the logical expression in the IF command is false:

Field:	PF (CUSTPH 1 3)	T.TITLE
Contents before:	'409'	'ZZZZZZZZZZZZZZZZZZZZ'
Contents after:	'409'	'OUTSIDE OF HOUSTON'

Because an ELSE command is coded, the program transfers control to the ELSE command and executes the procedure statements that follow.

For the same code with the following field values, the logical expression in the IF command is true:

	PF (CUSTPH 1 3)	T.TITLE
Contents before:	'713'	'OUTSIDE OF HOUSTON'
Contents after:	'713'	'HOUSTON'

The program processes the procedure statements following the IF command until it encounters the ELSE command. The program then passes control to the corresponding END command for the current IF command.

Example 2

```
DO CASE
  CASE AREACODE= '713'
    LET T.TITLE= 'HOUSTON'
  CASE AREACODE= '806'
    LET T.TITLE= 'NORTH WEST TEXAS'
  CASE AREACODE= '915'
    LET T.TITLE= 'WEST TEXAS'
  CASE AREACODE= '512'
    LET T.TITLE= 'SOUTH WEST TEXAS'
  CASE AREACODE= '409'
    LET T.TITLE= 'S.E. TEXAS (EXCEPT HOUSTON)'
  CASE AREACODE= '817'
    LET T.TITLE= 'NORTH CENTRAL TEXAS (FT. WORTH)'
  CASE AREACODE= '214'
```

```
        LET T.TITLE= 'N.E. TEXAS (INCLUDING DALLAS)'  
ELSE  
        LET T.TITLE= 'OUTSIDE OF TEXAS'  
END
```

If AREACODE is one of the Texas state area codes, the program assigns T.TITLE field to the portion of the state corresponding to the area code. If the area code is not one of the Texas state area codes, the program executes the ELSE command and sets the T.TITLE field to OUTSIDE OF TEXAS.

END Command

END [{DO | IF | PROC}]

The END command signifies the end of an IF, DO, or PROC block. If no block parameter is given, the END command is assumed to be for the innermost active block.

- DO Specifies that this END command is for the current DO block.
- IF Specifies that this END command is for the current IF block.
- PROC Specifies that this END command is for the current procedure block.

Examples.

```
DO FORALL 1.PARM
  IF 1.RECORD NE ' ' THEN
    LET T.FIELD1 = 1.FIELD1
    LET T.FIELD2 = 1.FIELD2
    LET T.FIELD3 = 1.FIELD3
  END IF
END DO
```

```
DO CASE
  CASE CLASS = 1
    LET RATE = RATE * 1.05
  CASE CLASS = 2
    LET RATE = RATE * 1.10
END
```

Specify an END command for every DO or IF command.

In the example, the first END command delimits the end of the embedded IF command. The second END command delimits the end of the DO FORALL command. The third END command delimits the end of the DO CASE command.

```
PROC MAXITEMS 25
  CALL PROC XYZ
END PROC
```

The above example shows the use of the END command to terminate a procedure block.

FIELD Command

```
field name:FIELD [TYPE] field type [[LENGTH] field-length]
                                [DECIMALS decimal-places]
                                [FLOAT floating-edit-char]
                                [FILL fill-edit-char]
                                [TRAIL trailing-edit-char]
                                [EDLEN edit-length]
                                [INIT initial-value]
                                [HEADING `line1` [`line2']]
```

The FIELD command defines a temporary field within a procedure. Once you define a temporary field in the FIELD command, you can use it in other procedures, reports, or subfiles. Make it a practice to define all temporary fields in the first procedure in the application. The statement label is the field name.

<i>field name:</i> FIELD FLD	<p>Specifies a temporary field name.</p> <ul style="list-style-type: none"> ■ Make field names from one to eight characters long starting with an alphabetic character. See Chapter 2, Terminology, Syntax, and Processing for valid field names. ■ Place a colon at the end of the field name followed by one or more blanks. ■ Even though the syntax appears to be “labeling” the line, the FIELD keyword causes the line to be a temporary field definition. ■ Specify all FIELD statements at the beginning of a procedure.
TYPE <i>field-type</i>	<p>Specifies the field type. Make this operand a 1-character code corresponding to the storage type of the field being defined (for example, C for character type field, Z for zoned decimal).</p> <p>For valid entries, see Appendix C, Technical Notes.</p>
LENGTH <i>field-length</i> LEN	<p>Specifies the field length. Use an integer constant, with a value within the range of the limits imposed by the field type, as the operand.</p> <p>For default field lengths for each field type, see Appendix C, Technical Notes.</p>
DECIMALS <i>decimal-places</i> DEC	<p>Specifies the number of decimal places for numeric fields. Use an integer from 0 through 9 as the operand. Make this value smaller than the length of the numeric field.</p>

FLOAT <i>floating-edit-char</i> FLT	<p>Specifies the floating edit character. Use a single character constant as the operand.</p> <p>For valid floating edit characters, see Appendix C, Technical Notes.</p>
FILL <i>fill-edit-char</i>	<p>Specifies filling edit character. Use a single character constant as the operand.</p> <p>For valid fill edit characters, see Appendix C, Technical Notes.</p>
TRAIL <i>trailing-edit-char</i> TRL	<p>Specifies the trailing edit character. Use a single character constant as the operand.</p> <p>For valid trailing edit characters, see Appendix C, Technical Notes.</p>
EDLEN <i>edit-length</i>	<p>Specifies an overriding edit length. Use an integer constant as the operand.</p> <p>For valid edit length entries, see Appendix C, Technical Notes.</p>
INIT <i>initial-value</i>	<p>Specifies an initial value. Use a constant, whose type corresponds to the field type (For example, for numeric type fields use numeric digits, as the operand. Add a negative sign and decimal point, if needed.)</p>
HEADING <i>'line1' 'line2'</i> HEAD	<p>Specifies one or two lines of column heading for the field if it is output to a report.</p> <ul style="list-style-type: none">■ Use character constants enclosed in single quotation marks as the operands.■ Make each line of heading up to 14 characters long.■ To force a blank column heading, insert the system delimiter after a blank within the quotation marks.■ Use two consecutive single quotation marks within a heading to represent a single quotation mark.■ If you leave this operand blank, the field name is used as the column heading.

Example 1

```
FRSTTIME: FIELD TYPE C LENGTH 1 INIT 'Y'
```

You can reference the temporary field by using the qualifier T in front of the field name (for example, T.FRSTTIME). The field contains character type values (such as A to Z, 0 to 9, blanks) and is one character long with an initial value of Y.

Example 2

```
BALANCE: FIELD Z 8 2 HEADING 'CUSTOMER' 'BALANCE'
```

The temporary field T.BALANCE contains numeric values (0 to 9, positive or negative), has two decimal places, and is eight digits long. The first six digits are to the left of the implied decimal place. If the temporary field is output to a report, the column heading:

```
CUSTOMER  
BALANCE
```

is centered over the column of data being printed.

GO TO Command

GO [**TO**] *jump-to-label*

The GO command branches forward to another labeled procedure statement in the current procedure. Create labels for procedure statements by starting the line with the label followed immediately by a colon and one or more blanks.

TO *jump-to-label* Specifies the procedure statement to be executed next. You can only jump to a subsequent labeled statement in the current procedure.

The colon used on the labeled procedure statement is not used on the GO TO command.

The GO command only branches forward to a subsequent labeled statement in the current procedure. Do not attempt to use the GO command to branch backward to a prior labeled statement.

You can use a GO statement to jump out of an IF-END or DO-END group of statements. If you use a GO in this manner, it overrides the WHILE, UNTIL, FOR, and FORALL conditions governing the DO loop.

Note: The periods mean that some of the procedure code has been omitted from the example.

Example 1

```
.  
. .  
GO TO LABEL10  
. .  
. .  
LABEL10: LET A = A + 1  
. .  
. .
```

The example above shows the syntax of the GO command branching to a labeled line forward in the procedure called LABEL10. The format of a label is shown on the procedure statement with the LET command. When the GO command is executed, all procedure statements between the GO command and the line labeled LABEL10 are bypassed and processing continues with the LET command on the labeled line.

Note: The periods between the END command and the ERROR labeled statement mean that some of the procedure code has been omitted from the example.

Example 2

```
IF VALIDATE(ORDRDATE DATE)  
  CALL REPORT ORDER  
ELSE  
  LET T.ERROR = ORDRDATE  
  LET T.MSG = 'INVALID ORDER DATE'
```

```
GO TO ERROR
END
;
IF VALIDATE (ITEMNO PATTERN P'9999999')
  CALL REPORT ITEM
ELSE
  LET T.ERROR = ITEMNO
  LET T.MSG = 'INVALID ITEM NUMBER'
  GO TO ERROR
END
.
.
.
ERROR: CALL REPORT ERROR
```

The label ERROR is in a CALL command for an error report. Only the first error detected is reported with this code.

- If the order date is an invalid date, T.ERROR and T.MSG are set to indicate the invalid date and a corresponding message. The next procedure statement is a GO TO command that branches forward in the procedure to the line labeled ERROR at the bottom of the procedure.
- If the order date is valid, the CALL command for the report ORDER is executed, and processing continues with the next IF command. If the field ITEMNO does not contain all numerics according to the pattern, T.ERROR and T.MSG are set to indicate the invalid item number and a corresponding message. The next procedure statement is a GO TO command that branches forward in the procedure to the line labeled ERROR at the bottom of the procedure.

IF Command

IF [CONDITION] *logical-expression* [THEN]

Place the IF command at the beginning of a group of statements (an IF block) to be performed when the logical expression on the IF command is true.

The program transfers control to the corresponding ELSE or END statement for this IF block when the logical expression is false.

Use an END command to delimit the IF block of procedure statements.

CONDITION *logical-expression*
COND Specifies the logical expression. The logical expression can contain:

- Arithmetic sub-expressions (see examples for further explanation).
- PF, LOOKUP, FIND, SCAN, and VALIDATE functions.

THEN Is an optional entry with no operands. Use it for clarity and readability.

Example 1

```
IF CUSTNO = '00001' THEN
  LET T.SET = 'ON'
ELSE
  LET T.SET = 'OFF'
END
```

When the field CUSTNO contains 00002, the program evaluates the logical expression on the IF command as false, passes control to the ELSE command within the IF block, and sets the field T.SET to OFF.

Example 2

```
IF PF(ORDRDATE 5 2) = PF(F.TODAY 5 2) -1 THEN
  CALL REPORT LASTYEAR
END
```

If the field ORDRDATE contains 011594 and the field F.TODAY contains 011595, the logical expression on the IF command is true.

The logical expression on the IF command has an arithmetic sub-expression PF(F.TODAY,5,2) -1. The last two positions of the flag field TODAY minus 1 are compared to the last two positions of ORDRDATE. This logic isolates the year portions of the two fields to see if the order was made last year with respect to the current year.

Example 3

```
IF SCAN(CUSTNAME FOR 'ACME') THEN
  CALL REPORT COMPANY
END
```

The SCAN function serves as the logical expression, because it is a conditional function. The program searches for the characters ACME in the field CUSTNAME.

- If the program finds the character string, the SCAN function is true and the program executes the CALL command for the report COMPANY.
- Otherwise, if the logical expression is false, the program bypasses the procedure statements between the IF command and the END command and passes control to the statement following the END command.

INCLUDE Command

```
INCLUDE [ITEM] item-name,  
        [DATEFMT {DATE | TODAY | TODAYX | ISDATE | JULIAN | JULANX | mmddy}],  
        [INFO 'text']
```

The INCLUDE command is used to include a cataloged request into the application. This command may not be embedded inside a procedure but must be placed either before or after a procedure block to include a cataloged request.

ITEM <i>item-name</i>	Specifies the name of the cataloged request or request group that is to be included in the application.
DATEFMT <i>format-code</i> DFMT	Specifies a code indicating how the report date is to be formatted or specifies the date to be used as the report date. Valid format codes are as follows: DATE -- Specifies that the date be formatted as mmm dd, yyyy (e.g. DEC 25, 2001) according to the country conventions specified at installation time in M4PARAMS. TODAY -- Specifies that the date be formatted as mm/dd/yy (e.g. 12/25/01) according to the county conventions specified at installation time in M4PARAMS. TODAYX -- Specifies that the date be formatted as mm/dd/yyyy (e.g. 12/25/2001) according to the country conventions specified at installation time in M4PARAMS. ISDATE -- Specifies that the date be formatted as yyyy-mm-dd (e.g. 2001-12-25). JULIAN -- Specifies that the date be formatted as yyddd where ddd is the serial day number of the year (e.g. 01359). JULANX -- Specifies that the date be formatted as yyyyddd where ddd is the serial day number of the year (e.g. 2001359).

Alternatively, the date that is to be used as the report date may be specified as mmddy according to the country conventions specified at installation time in M4PARAMS (e.g. 070402 would be displayed as 07/04/02 using USA conventions).

If the DATEFMT keyword is omitted, the TODAY format will be assumed.

INFO *'text'*

Specifies descriptive text that may be used to annotate the cataloged request. The text cannot be longer than 28 characters.

Examples

```
INCLUDE CATREQ1
```

The above example shows the use of the INCLUDE command to include cataloged request CATREQ1 in the application.

```
PROC  
  CALL STATREPT  
END PROC  
INCLUDE STATREPT, DATEFMT TODAYX, INFO 'Status Report'
```

The above example shows the use of the INCLUDE command to include a cataloged request that is a subroutine request along with the code to call the cataloged request.

LEAVE Command

LEAVE

The LEAVE command exits the corresponding DO block immediately (that is, transfers control to the statement after the END command for the current block).

If the LEAVE command is within a nested DO block, the exit is to the outer DO block. See Example 3 on the following page.

The LEAVE command can only be used in DO blocks. The LEAVE command cannot be used in DO blocks with the keyword of CASE. The LEAVE command has no keywords.

Example 1

```
LET T.COUNT = 0
DO UNTIL T.COUNT = 1000
  IF FIND(SEGMENT 1.ROOT)
    LET T.TOTAL = T.TOTAL + 1.AMOUNT
    LET T.COUNT = T.COUNT + 1
  ELSE
    LEAVE
  END
END
```

In this example, the LEAVE command transfers control out of the DO block if there are less than 1000 records on the file being read by the FIND function. If the file being read on the FIND function runs out of records before the DO UNTIL command is satisfied, the false condition in the IF command passes control to the ELSE command where the LEAVE command is executed. Without the LEAVE command, there would be no way to exit the DO-END set of statements.

Example 2

```
LET T.ORDRCNT = 0
DO FORALL ORDER
  IF ORDCMPLT = 'Y' THEN
    LET T.ORDRCNT = T.ORDRCNT + 1
  ELSE
    CALL REPORT ERROR
    LEAVE
  END
END
```

This section of code counts all of the completed orders. If any order is not complete (that is, ORDCMPLT is not Y), the program executes the ELSE command along with the subsequent procedure statements. The program processes the report ERROR and exits the DO block without processing all of the ORDER segments.

Example 3.

```
DO UNTIL T.COUNT = 1000
  DO FORALL SEG20
    .
    .
    . LEAVE
    .
    .
  END
END
```

The LEAVE command is within the inner DO block and when it is executed, the program passes control to the statement after the END command which corresponds to the inner DO block. The outer DO block is not affected by the LEAVE command.

LET Command

```
LET [FIELD] result-field = source-expression [WITH] [EDIT P'pattern']  
[ROUNDING] [JUSTIFY {LEFT | RIGHT}]
```

Note: The IBM Language Environment (LE) service, CEEDATE, is Year 2000 compliant.

The LET command replaces or moves data between the source-expression and the result-field.

FIELD <i>result-field</i> FLD	Specifies the receiving field. Use a field name or PF function as the operand.
source-expression	Specifies the “from” field or arithmetic expression. Use a name, constant, or arithmetic expression, that can include any value function (PF or LOOKUP), as the operand.
WITH	Is an optional entry with no operands. Use it for clarity and readability where needed.
EDIT P' <i>pattern</i> '	Specifies the edit pattern that is used to edit the result-field. Specify a pattern string starting with the letter P, followed by a beginning single quotation mark, a string of special pattern symbols, and ending with a closing quotation mark. It can be up to 31 characters long. Use EDIT only if the result-field is a character string. For valid edit pattern symbols, see Appendix C. Technical Notes . If the source-expression is a date field, the edit pattern is a date format picture, for example MM/DD/YY. For a list of valid picture characters. Use the date format picture as a parameter in a call to CEEDATE, see the <i>IBM Language Environment Reference Manual</i> .
ROUNDING	Specifies that the result-field for all arithmetic and replacement operations should contain rounded results. Note that with the possible exception of divide, no rounding occurs if the number of decimal places in the result-field is not less than the number of decimal places in the computed or replacement value.

JUSTIFY {LEFT |
RIGHT}

Specifies that the result is to be left-justified or right-justified after the move of the data. The direction operand must be LEFT or RIGHT to indicate the direction in which the result-field is to be justified.

- If you omit RIGHT or LEFT, no action beyond replace is taken.
- JUSTIFY can only be used if the result-field is a character string.
- If EDIT and JUSTIFY are both specified, the JUSTIFY take places after the edit.

Example 1

```
LET T.NUMBER = 1.5
```

For the following field values, the temporary field T.NUMBER is replaced with the numeric constant on the right side of the equal sign:

Field:	T.NUMBER
Contents before:	'0.0'
Contents after:	'1.5'

Example 2

```
LET FIELD1 = FIELD2 JUSTIFY LEFT
```

For the following field values, the contents of field FIELD2 are moved into the field FIELD1:

Field:	FIELD1	FIELD2
Contents before:	'ZZZZZ'	' ABC'
Contents after:	'ABC '	' ABC'

The keyword JUSTIFY specifies LEFT for the direction and left-justifies the contents of the FIELD1 field at the end of the LET command.

Example 3

```
LET A = X*Y+10
```

For the following field values, the contents of field X are multiplied by the contents of field Y, 10 is added, and the result is stored in field A:

Field:	A	X	Y
Contents before:	'0'	'5'	'5'
Contents after:	'35'	'5'	'5'

Example 4

LET T.AMOUNT = PAY + 100 WITH EDIT P' \$, \$\$\$.99'

For the following field values, the contents of field PAY plus 100 are stored in the field T.AMOUNT with the specified edit pattern:

Field:	T.AMOUNT	PAY
Contents before:	' 0'	' 950.25'
Contents after:	' \$1,050.25'	' 950.25'

Example 5

LET RESULT = LOOKUP (MONTHS ARGUMENT ORDERMM)

This example uses the following lookup table where a number represents a month.

MONTHS	(table)
01	JANUARY
02	FEBRUARY
03	MARCH
04	APRIL
05	MAY
06	JUNE
07	JULY
08	AUGUST
09	SEPTEMBER
10	OCTOBER
11	NOVEMBER
12	DECEMBER

	RESULT	ORDERMM
Before:	' ZZZZZZZZ'	' 05'
After:	' MAY '	' 05'

The program “looked up” value of the field ORDERMM in the argument list of the MONTHS table. When the program finds a match, the program uses the result value from the table as the value of the LOOKUP function and, in this case, puts it into the field RESULT.

Example 6

```
LET PF(T.STRING 02 04) = T.MIDDLE
```

For the following field values, the LET command moves the contents of the field T.MIDDLE into the field T.STRING starting in location 2 for a length of 4:

Field:	T.STRING	T.MIDDLE
Contents before:	'123456789'	'ABCD'
Contents after:	'1ABCD6789'	'ABCD'

LOCATE Command

Note: LOCATE is available in *VISION:Builder* and *VISION:Two*, but not *VISION:Inform*.

```
LOCATE [ARRAY] array-identifier
      { [ROW row-number] [COLUMN column-number] }
```

LOCATE LOC	This command processes arrays by locating a particular data cell in the array, a particular row in the array, or a particular column in the array.
ARRAY <i>array-identifier</i> ARR	Specifies the array-identifier. Use an identifier that identifies an array in the current application as the operand.
ROW <i>row-number</i>	Specifies the row of the array. Use a constant, field name, or arithmetic expression, whose value specifies the row number to locate, as the operand.
COLUMN <i>column-number</i> COL	Specifies the column of the array. Use a constant, field name, or arithmetic expression, whose value specifies the column number to locate, as the operand.

- If you use both ROW and COLUMN, the program locates a specific data cell in the array.
- If you code only ROW, the program processes all data cells in the row specified.
- If you code only COLUMN, the program processes all data cells in the column specified.

The program sets the system flag F.COLUMN to the column number being processed as a result of the LOCATE command and sets the system flag F.ROW to the row number being processed.

```
Array-identifier:  A
Array name:       BIRTHDAY
```

	COLUMN 1	COLUMN 2
ROW1	JOE 020359	KAREN 122551
ROW2	DOUG 050658	SARA 100960

Example 1

```
LOCATE ARRAY A ROW 2 COLUMN 1
```

The program finds the data cell with DOUG and 050658 as a result of the LOCATE command. When you use the field name(s) for the fields in the data cell, the program processes only the one data cell that is located. At the end of the procedure, the program automatically releases the array. The RELEASE command can be performed before the end of the procedure to allow processing of the entire array again.

Example 2

```
LOCATE A ROW X+1
```

If the field X contains 1, the arithmetic expression results in row 2 being located (F.ROW contains 2). When fields in the array are referenced after the LOCATE command, only row 2 is processed, looping through all columns in turn. F.COLUMN contains each column number as the column is being processed.

Example 3

```
LOCATE A COLUMN T.COL
```

If the field T.COL contains 1, the value of the temporary field results in column 1 being located (F.COLUMN contains 1). When fields in the array are referenced after the LOCATE command, only column 1 is processed, looping through all rows in turn. F.ROW contains each row number as the row is being processed.

PROC Command

```
[proc-name:] PROC [TYPE {NORMAL | SUBROUTINE | INIT | PRE_MASTER_READ |  
TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |  
EOF | EOFPLUS}],  
  
[TEMPREINIT],  
[SELECTIONCONTROL {YES | NO}],  
[MAXITEMS number],  
[BACKBRANCH number],  
[INFO 'text'],  
[PARALLEL_LOOPING] (use with caution)
```

The PROC command is used to begin a new procedure block. A procedure block may contain any procedural command as well as report blocks and extract statements. When the PROC command is used, no fixed syntax ER statement is needed nor are the ##PROC and ##PEND statements required to bracket the ASL procedural commands.

<i>proc-name</i> :	Specifies the name of the procedure. Naming a procedure is only required when the procedure TYPE is specified as SUBROUTINE.
--------------------	--

TYPE *invocation-event*

Specifies the event within the processing cycle at which the procedure will be invoked. If this keyword is omitted, the procedure will be invoked during each iteration of the master file read and standard coordination cycles. Valid invocation-event specifications are:

NORMAL -- Specifies that the procedure will be invoked during each iteration of the master file read and standard coordination cycles.

SUBROUTINE or SUB -- Specifies that the procedure will only be invoked when explicitly called by a procedural CALL statement.

INIT -- Specifies that the procedure will be invoked only one time after the master and coordinated files have been opened but before any records from these files have been processed.

PRE_MASTER_READ -- Specifies that the procedure will be invoked just prior to reading the next master file record.

TYPE1 -- Specifies that the procedure will be invoked after each transaction record has been read.

TYPE2 -- Specifies that the procedure will be invoked after the transaction file and the master files have been aligned.

TYPEM -- Specifies that the procedure will be invoked after the master file record has been updated but before first round coordination.

TYPE3 -- Specifies that the procedure will be invoked after the master file record has been updated but after first round coordination.

TYPE4 -- Specifies that the procedure will be invoked whenever a transaction record is rejected, either explicitly by procedural code or implicitly by transaction validation controls.

EOF -- Specifies that the procedure will be invoked only once after the transaction file, master file, and all coordinated files have reached end of file.

EOFPLUS -- Specifies that the procedure will be invoked as in NORMAL and additionally as in EOF above.

TEMPREINIT

REINIT

Specifies that temporary fields explicitly defined in this procedure are to be re-initialized every time this procedure is invoked. If this keyword is omitted, the contents of any temporary fields defined in this procedure will be unchanged from their last usage when this procedure is invoked.

SELECTIONCONTROL {YES | NO}

SELCNTL	<p>Specifies whether or not all occurrences of repeated segments in a hierarchical structure are to be examined.</p> <p>YES -- Specifies that only the first occurrence of a repeated segment within each record that meets the selection qualification will be selected.</p> <p>NO -- Specifies that all occurrences of repeated segments within each record that meet the selection qualification will be selected. This is the default if the SELECTIONCONTROL keyword is omitted.</p>
MAXITEMS <i>number</i>	<p>Specifies the maximum number of records that will be selected for this procedure. Specify a number from 1 to 9999 to limit the number of records that will be selected for this procedure. If this keyword is omitted, the number of records that will be selected is unlimited.</p>
BACKBRANCH <i>number</i> BACKB	<p>Specifies the maximum number of times that a TRANSFER command to a procedure earlier on in the transaction processing cycle may be executed.</p>
INFO ' <i>text</i> '	<p>Specifies descriptive text that may be used to annotate the procedure. The text cannot be longer than 28 characters</p>

PARALLEL_LOOPING

This is a deprecated specification that modifies the automatic looping algorithm for hierarchical structures containing multiple branches such that the Cartesian product of all child occurrences within a parent will not always be produced. Although PARALLEL_LOOPING may optimize performance, processing and output results will be data dependent and therefore, could be misleading. The ABBREVIATED keyword specification on the REPORT and EXTRACT commands is the preferred specification to eliminate redundant output when dealing with multi-branch hierarchical structures or views. Use of the PARALLEL_LOOPING specification is strongly discouraged and should only be used in specific situations where the implications of its use are fully understood.

Examples

```
MAIN: PROC INFO 'Main Procedure'  
PROC SELECTIONCONTROL YES, MAXITEMS 500  
SUB1: PROC TYPE SUBROUTINE
```

RELEASE Command

RELEASE {[SEGMENT] *segment-name* | ARRAY *array-identifier*}

RELEASE REL	Releases a segment specified by the FIND function or an array specified by the LOCATE command. A FIND or LOCATE command limits the application's view of data that is available. Subsequent field references process data only from segments that result from a FIND command or cells that result from a LOCATE command. The RELEASE command unlocks the application view and makes available all segments, or cells, to subsequent statements. After the RELEASE command in the procedure, any reference to the segment or array results in processing, beginning with the first occurrence of the segment or the beginning of the array. Note: The RELEASE command ARRAY keyword is available in <i>VISION:Builder</i> and <i>VISION:Two</i> , but not <i>VISION:Inform</i> .
SEGMENT <i>segment-name</i> SEG	Specifies a segment name to release. Do not use the name of a root segment as the operand.
ARRAY <i>array-identifier</i> ARR	Specifies the identifier of an array to release. Use an identifier corresponding to the array to be released as the operand.

Example 1

```
IF FIND(SEGMENT ORDER WHERE ORDERNO = 10001)
  RELEASE SEGMENT ORDER
  CALL REPORT ORDERS
END
```

The FIND function is true if there is at least one segment with an order number of 10001. The function positions on the first segment that is found.

The RELEASE command releases the ORDER segment that was found and the report outputs all of the current orders. Without the RELEASE command, the report ORDERS would only have access to the single segment established by the FIND function.

Example 2

The Birthday array from the examples for the LOCATE command is being used in this example.

```
LOCATE ARRAY A ROW 1 COLUMN 1
IF A.NAME EQ 'JOE 020359' THEN
  CALL REPORT NAMES
ELSE
  RELEASE ARRAY A
END
```

First, the program locates a cell of the array with the LOCATE command giving the ROW and COLUMN of the data cell. The program tests the field A.NAME in the array and, if the field does not contain JOE 020359, executes the ELSE command and releases the array. Otherwise, the data cell located at row 1 column 1 is the only cell available after the END of the IF block.

REPLACE Command

REPLACE [**STRING**] *search-string* [**IN**] *modify-field* [**WITH**] *substitute-value*

REPLACE
REP This command replaces all occurrences in *modify-field* of *search-string* with *substitute-value*.

- The scan of the *modify-field* for the *search-string* begins at the leftmost character in the *modify-field*.
- The *substitute-value* is substituted into the *modify-field* in place of the *search-string*.
- The scan of the *modify-field* for the *search-string* then resumes starting with the first character after the substituted characters.

This process continues until the end of the scanned field is reached.

STRING *search-string*
STR Specifies the string to search for in *modify-field*. The operand can be a field name, character constant, or validation pattern.

For valid pattern symbols, see [Appendix C, Technical Notes](#).

IN *modify-field* Specifies the field to be modified (that is, the field that contains the *search-string*). Use a field name or a PF function as the operand.

WITH *substitute-value* Specifies the value to be substituted into *modify-field* instead of *search-string*. Use a field name or character constant as the operand.

The system field **SSCOUNT** counts the number of matches found during the **REPLACE** operation.

- If the *search-string* is longer than the *modify-field*, the **F.SSCOUNT** system field is set to zero.
- If the *modify-field* or the *search-string* is null (an empty variable length field) or invalid and/or the *substitute-value* is invalid, the operation is not performed and the **F.SSCOUNT** system field is set to -2.
- If the *search-string* and the *substitute-value* are of different lengths, the **REPLACE** command is executed according to the following rules:
 - If the *search-string* is smaller than the *substitute-value*, the portion to the right of the matching characters is shifted to the right to make room for the characters to be substituted.
Non-blank characters might be lost on the right (truncated) by this process, unless the *modify-field* is a variable length field.

If the substitution would cause the maximum length of a variable length field to be exceeded, the operation does not take place, the modify-field becomes invalid, and the system field F.SSCOUNT is set to a -1.

- If the search-string is larger than the substitute-value, the portion of the modify-field to the right of the matching characters is left-justified and placed adjacent to the substituted characters.
If the modify-field is a fixed length character field, trailing blanks are created in the field.

If the substitute-value is a variable length field with a null length, a zero length field is substituted for the search-string, effectively eliminating it and left-justifying the remaining portion of the field.

Example 1

```
REPLACE 'ABC' IN CUSTNAME WITH 'XYZ'
```

For this example, each time the literal ABC is found in the field CUSTNAME, it is replaced with the literal XYZ. F.SSCOUNT is set to 1, because one match was found:

```
Field:                CUSTNAME
Contents before:      'THE ABC COMPANY'
Contents after:       'THE XYZ COMPANY'
```

Example 2

```
REPLACE P'ZZZ999' IN ITEMNO WITH '111QQQ'
```

For this example, each time the pattern of three alphabetic characters followed by three numerics is found in the field ITEMNO, it is replaced by the literal 111QQQ. F.SSCOUNT is set to 1 because one match was found:

```
Field:                ITEMNO
Contents before:      '3AAA222C'
Contents after:       '3111QQQC'
```

Example 3

```
REPLACE '/' IN SHIPDATE WITH T.NULL
```

For this example, each time the program finds '/' in the field DATE, the program substitutes the null value from the field T.NULL. The program sets F.SSCOUNT to 2, because two matches were found:

```
Field:                SHIPDATE                T.NULL (TYPE V, EMPTY)
Contents before:      '01/15/01'                ' '
Contents after:       '011501 '                ' '
```

RETURN Command

RETURN
RET

Use the RETURN command in a subroutine procedure to branch back to the calling procedure. Note that a RETURN statement is not required for a subroutine procedure to get back to the calling procedure, because the end of a procedure constitutes an automatic RETURN.

Control automatically passes back to the calling procedure after all of the statements in the subroutine procedure process.

Example

```
IF ORDCMPLT = 'Y'  
  CALL REPORT ORDERS  
  RETURN  
END  
;  
IF DATE LE '010115'  
  CALL REPORT TRACKDWN  
  RETURN  
END  
CALL REPORT ALLOTHRS
```

These procedure statements make up a subroutine procedure that you can CALL from any other procedure. When the subroutine procedure executes the RETURN commands, the subroutine procedure passes control back to the calling procedure.

If ORDCMPLT is equal to Y or the DATE is less than or equal to 010115, the program produces a report line and immediately exits the subroutine procedure by one of the RETURN commands. The subroutine procedure automatically exits after processing the CALL command to report ALLOTHRS where the RETURN is implied.

TRANSFER Command

TRANSFER [TO] {NEXT_MASTER | TYPE_1 | TYPE_2}

The TRANSFER command exits procedures immediately and transfers processing control to another type of procedure.

NEXT_MASTER Causes the system to read the next master file record and start the processing cycle again.

TYPE_1 Transfers control to the Transaction Record Initial Validation control procedure. Because this procedure has already been executed, a TRANSFER TYPE_1 constitutes recycling to an earlier point in the processing cycle. The results of all previous procedures (including the prior execution of the event-controlled procedure) are still in effect.

TYPE_2 Transfers control to the Transaction/Master File Alignment control procedure. Because this procedure has already been executed, a TRANSFER TYPE_2 constitutes recycling to an earlier point in the processing cycle. The results of all previous procedures (including the prior execution of the event-controlled procedure) are still in effect.

Example

```
IF CUSTOMER LT '01000'  
  TRANSFER NEXT_MASTER  
END
```

In this example, the program processes only CUSTOMER values greater than or equal to 01000 through the remainder of the procedure after the END command. For CUSTOMER values less than 01000, the TRANSFER command is executed. The program reads the next master file and then starts the processing cycle again.

Report Command Group

The Report command group includes the commands that are used to specify report output. The statements used to specify report output must begin with a REPORT command and end with an END command. These statements constitute a report block.

A report block may in turn contain one or more groups of statements that specify the layout for a section of the report (page title, column headings, or summaries) in a formatted report. These statements are defined as section block commands.

The following table identifies each of the commands in the Report command group along with an indication of whether the command is restricted for use within the section block:

Command	Function	Section Block Only
AVERAGE	Compute an average summary	
COMPUTE	Compute a value in a Summary Section of a formatted report	Yes
COUNT	Compute a count summary	
CUMULATE	Compute a cumulative total summary	
DATA	Format data for a line in a formatted report	Yes
END	End of Report Block or Section Block	
FORMAT	Report formatting parameters and controls	
GROUP	Report data grouping controls	
ITEM	Report data content and controls	
LINE	Format and output data line in a formatted report	Yes
MAX	Compute a maximum value summary	

Command	Function	Section Block Only
MIN	Compute a minimum value summary	
NEWPAGE	Control the beginning of a new page in a formatted report	Yes
ORDER	Report data sorting controls	
PERCENT	Compute a percent summary	
PREFACE	Report preface page contents	
RATIO	Compute a ratio summary	
REPORT	Begin report and specify report content	
SECTION	Begin a section of a formatted report	Yes
SIZE	Control the size of the detail section in a formatted report	Yes
SKIP	Skip lines (leave blank) in a formatted report	Yes
TITLE	Report title line contents	
TOTAL	Compute a total summary	
XREP	Extended report controls	

A report block may be embedded inside a procedure block, and a procedure block may contain more than one report block. When embedded inside a procedure, a report will be invoked based upon the logic flow of the procedure. When present outside of a procedure, a report will be invoked based upon the TYPE keyword.

A section block must always be inside of a report block. Only one of each of the three different section types may occur inside of a report block.

Partial field notation may be used for most character type fields referenced within a report block. The statement descriptions explicitly note those instances where partial field notation is not accepted. Unlike varchar field usage in procedure block statements, partial field notation is not allowed for varchar fields referenced within a report block.

AVERAGE Command

```
AVERAGE [ITEM] summary-field ...,  
        {BY group-field | AT LEVEL level-number}
```

The AVERAGE command is used to specify that an average type summary is to be computed for the field. AVG may be used as an abbreviation for AVERAGE.

ITEM <i>summary-field</i> ...	Specifies the qualified names of one or more fields for which an average summary is to be computed. The field may be a partial field using the PF function notation.
BY <i>group-field</i>	Specifies the qualified name of the field by which the summaries are to be grouped. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the AT LEVEL keyword.
AT LEVEL <i>level-number</i>	Specifies a control break level number by which the summaries are to be grouped. Valid specifications for level number are 1 through 9 and G. If this keyword is specified, a GROUP command with an AT LEVEL keyword must also be specified. The GROUP command specifies the name(s) of the field(s) by which the summaries are grouped. This keyword is mutually exclusive with the BY keyword.

Examples

```
AVERAGE AMOUNT BY CUSTNO
```

Compute the average of the field named AMOUNT grouped by CUSTNO.

```
AVERAGE SALARY AT LEVEL 2
```

Compute the average of the field named SALARY at the level 2 control break.

COMPUTE Command

```
COMPUTE STEMPnn = operand1 operator operand2,  
    [PICTURE P'picture'],  
    [DECIMALS decimal-places],  
    [ROUNDED],  
    AT LEVEL level-number
```

The COMPUTE command is used to compute special temporary values to be used in a summary section. The result of the computation is stored in an STEMP field named STEMPnn where nn is specified as 01-99 to uniquely identify the field. A maximum of 99 STEMP fields may be defined in this way. Note that the AT LEVEL specification is always required.

STEMPnn	Specifies the result field for the command. The nn of STEMPnn uniquely identifies the STEMP result field and may be a value between 01-99 (leading 0 required when the value is less than 10). STEMP fields defined with the COMPUTE command may be referenced in subsequent COMPUTE commands and output using the DATA and LINE commands.
---------	--

= operand1 operator operand2

Specifies the operation to be performed and the operand(s) to be used. If only operand1 is specified, the operation is a simple assignment of operand1 to the STEMP result. Otherwise, a simple arithmetic expression may be specified. The operator may be +, -, *, / representing addition, subtraction, multiplication, and division, respectively. Operand1 and operand2 may be STEMP fields, constants, or summary functions. STEMP fields appearing as operand1 or operand2 must have been defined as a result field in a previous COMPUTE command. Summary functions are specified as summary-function(field-name break-level) where summary-function may be TOTAL[TOT], CUMULATE[CUM], COUNT[CNT], MAX, MIN, AVERAGE[AVG], PERCENT[PCT], or RATIO[RTO] and break-level is the control break level (1-9 or G) at which the summary was computed. This level number must be at the same or a more minor level (higher number) than the number specified with the AT LEVEL keyword.

PICTURE P'*picture*'
PIC

Specifies the edit picture to be used when displaying the STEMP field. If this STEMP field is not displayed, this keyword is ignored. If this keyword is omitted on this command but was specified on a previous COMPUTE command for the same STEMP field, the previous picture will persist. See Appendix B for valid edit picture symbols.

DECIMALS *decimal-places*
DEC

Specifies the number of decimal places held in the result field. The value may be a number between 0 and 9.

ROUNDED

Specifies that rounding is to be performed when the STEMP result contains fewer decimal places than the operand(s) on the right side of the =.

AT LEVEL *level-number*

Specifies the control break level number (1-9 or G) at which this COMPUTE command is to be performed. The level number must have been specified or implied in a GROUP command. This specification may not be omitted from the command.

Example

```
COMPUTE STEMP01 = TOTAL(SALARY 2) * 1.10,  
PIC P'$$$,$$9.99', DEC 2, AT LEVEL 2
```

Compute an STEMP field named STEMP01 containing 2 decimal places using the total of SALARY at level multiplied times the constant 1.10. Display the value using the edit picture and perform the computation at the level 1 control break.

```
COMPUTE STEMP02 = STEMP01, DEC 0, ROUNDED, AT LEVEL 2
```

Compute the STEMP field named STEMP02 with 0 decimal places by rounding field STEMP01.

```
COMPUTE STEMP03 = TOTAL(SALARY 2) / TOTAL(SALARY 1),  
AT LEVEL 1  
COMPUTE STEMP03 = STEMP03 * 100
```

Compute the percentage that the total of SALARY computed at the last level 2 control break is of the total of SALARY at the level 1 control break.

COUNT Command

```
COUNT [ITEM] summary-field ...,  
      {BY group-field | AT LEVEL level-number}
```

The COUNT command is used to specify that a count type summary is to be computed for the field. CNT may be used as an abbreviation for COUNT.

ITEM <i>summary-field</i> ...	Specifies the qualified names of one or more fields for which a count summary is to be computed. The field may be a partial field using the PF function notation.
BY <i>group-field</i>	Specifies the qualified name of the field by which the summaries are to be grouped. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the AT LEVEL keyword.
AT LEVEL <i>level-number</i>	Specifies a control break level number by which the summaries are to be grouped. Valid specifications for level number are 1 through 9 and G. If this keyword is specified, a GROUP command with an AT LEVEL keyword must also be specified. The GROUP command specifies the name(s) of the field(s) by which the summaries are grouped. This keyword is mutually exclusive with the BY keyword.

Examples

```
COUNT ORDERNO BY CUSTNO
```

Compute a count of the ORDERNO items grouped by CUSTNO.

```
COUNT ORDERNO AT LEVEL 2
```

Compute the count of the ORDERNO items at the level 2 control break.

CUMULATE Command

```
CUMULATE [ITEM] summary-field ...,  
        {BY group-field | AT LEVEL level-number}
```

The CUMULATE command is used to specify that a cumulative total type summary is to be computed for the field. A cumulative total represents an accumulation of field totals since the beginning of the report. CUM may be used as an abbreviation for CUMULATE.

ITEM <i>summary-field</i> ...	Specifies the qualified names of one or more fields for which a cumulative total summary is to be computed. The field may be a partial field using the PF function notation.
BY <i>group-field</i>	Specifies the qualified name of the field by which the summaries are to be grouped. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the AT LEVEL keyword.
AT LEVEL <i>level-number</i>	Specifies a control break level number by which the summaries are to be grouped. Valid specifications for level number are 1 through 9 and G. If this keyword is specified, a GROUP command with an AT LEVEL keyword must also be specified. The GROUP command specifies the name(s) of the field(s) by which the summaries are grouped. This keyword is mutually exclusive with the BY keyword.

Examples

```
CUMULATE AMOUNT BY CUSTNO
```

Compute the cumulative total for AMOUNT grouped by CUSTNO.

```
CUMULATE SALARY AT LEVEL 2
```

Compute the cumulative total for SALARY at the level 2 control break.

DATA Command

```
DATA {'text' | LITERAL('text', repeat-count) | field-name |,  
      summary-function(field-name [level]) |,  
      STEMPnn | COL(column-number) | SPACES(number-spaces)}...,  
      AT LEVEL level-number
```

The DATA command is used to specify the content and layout for a portion of, or all of, a report line within a summary section (see SECTION command). Note that the DATA command does not output the report line. A subsequent LINE command is required to output the report line.

<i>'text'</i>	Specifies literal text that is to be placed into the report line at the current position.
LITERAL('text', repeat-count) LIT	Specifies literal text that is to be placed into the report line at the current position and repeated repeat-count times.
field-name	Specifies the qualified name of a field whose contents are to be placed into the report line at the current position. The field may not specify partial field (PF) notation.

summary-function(field-name [level])

Specifies that a summary value for the field identified by field-name and whose summary type is specified by summary-function is to be placed into the report line at the current position. Summary-function may be specified as any of the TOTAL[TOT], CUMULATE[CUM], COUNT[CNT], MAX, MIN, AVERAGE[AVG], PERCENT[PCT], or RATIO[RTO] summaries. When summary-function is specified, a corresponding summary command (for example, TOTAL command) or a REPORT command ITEMS operand summary function (for example, TOTAL(field-name BY field-name)) must also be specified. The summary-function level number operand specifies the control break level of the summary to be used and is optional. If specified, it must be a number between 1 and 9 or G and must be at a level equal to or lower (numerically higher) than the level number specified by AT LEVEL. If the summary-function level is not specified, it will default to the level number specified with the AT LEVEL keyword. Partial field notation (PF function) may not be used for the field.

STEMPnn

Specifies the STEM field that is to be placed into the report line at the current position. The STEM operand may only be used when the DATA command is part of a summary section. STEM fields must have been defined with a COMPUTE command prior to the DATA command. Refer to the COMPUTE command for further detail regarding the definition and use of STEM fields.

<code>COL(<i>column-number</i>)</code>	Specifies that the current columnar position within the print line is to be set to <i>column-number</i> .
<code>SPACES(<i>number-of-spaces</i>)</code>	Specifies that the current columnar position within the print line is to be advanced by <i>number-of-spaces</i> positions.
<code>AT LEVEL <i>level-number</i></code>	Specifies the control break level number at which the respective data elements are to be placed into the report line. Valid specifications for the number are 0 through 9 and G. A level number of 0 indicates that the data is formatted regardless of what the current control break level is set to. A level number of 1 to 9 or G (grand level) specifies that the data is formatted only when a control break has occurred at a level equal to or higher (numerically lower) than the number. The AT LEVEL keyword is required for every DATA command.

Examples

```
DATA COL(10) 'State Total for ' T.STATE SPACES(5),  
TOTAL(AMOUNT) AT LEVEL 1
```

The following line illustrates what the report line would contain beginning in column 10 for this Example

```
State Total for CA      $235,768
```

```
DATA LITERAL('-', 80) AT LEVEL G
```

A string of dashes 80 characters long would be placed into the print line when the control break is at the Grand level.

END Command

END [{REPORT | SECTION}]

The END command is used to signal the end of the current report block or section block within a report block. If no block parameter is given, the END command is assumed to be for the current innermost active block.

REPORT Specifies that this END command is for the current REPORT block.

SECTION Specifies that this END command is for the current SECTION block.

Examples

```
REPORT HIREDATE EMPNO LASTNAME FIRSTNME MIDINIT
ORDER BY HIREDATE, DESC HIREDATE
SECTION PAGETITLE
  LINE 'Report listing employees as of ' F.DATE,
      ' by hire date'
END SECTION
END REPORT
```

The above example shows the use of the END command for a section block and a report block.

FORMAT Command

```

FORMAT [HEIGHT number],
      [WIDTH number],
      [DATEPOS {UL | UR | MT | LL | LR | MB | ND}],
      [PAGEPOS {UL | UR | MT | LL | LR | MB | NP}],
      [TITLEPOS {TOP | BOTTOM}],
      [HEADINGS {YES | NO | NAME}],
      [HEADPOS {ABOVE | BELOW}],
      [DATEFMT {DATE | TODAY | TODAYX | ISDATE | JULIAN | JULANK |
               mmddy}],
      [BORDER {YES | NO | 'x'}],
      [STARTPAGE {number | PAGE}],
      [MAXPAGES number],
      [LINESPERPAGE number],
      [DETAILSPACING number],
      [INCOMPLETESUM 'x'],
      [NODATA {SKELETON | NOREPORT}],
      [SUBTITLE {REPEAT | NOREPEAT | NEWPAGE}],
      [SUMMARYLABELS {SPACE | NOSPACE | SUPPRESS}],
      [LINENUMS {NONE | LEFT | RIGHT | BOTH}],
      [IMAGES number [IMGTITLE {LOGPAGE | PHYPAGE | NEWPAGE}]]
      [METHOD {STDLIST | ALTLIST | CSV | TAB | HTML | PLAINTEXT |
              RAWDATA}],
      [STYLE number],
      [DDNAME ddname],
      [SUMFILE ddname],
      [AUTODEF]

```

The FORMAT command specifies various report parameters and controls.

HEIGHT <i>number</i>	Specifies the height of the printed report page, including margins, as a number of lines. If this keyword is omitted, the default page height specified at installation time in M4PARAMS is used.
WIDTH <i>number</i>	Specifies the width of the printed report page, including margins, as a number of characters.

DATEPOS position-code
DPOS

Specifies the position on the page at which the report date is to be placed. The valid position codes are as follows:

- UL -- Specifies the upper-left part of the page
- UR -- Specifies the upper-right part of the page
- MT -- Specifies the middle top part of the page
- LL -- Specifies the lower-left part of the page
- LR -- Specifies the lower-right part of the page
- MB -- Specifies the middle bottom part of the page
- ND -- Specifies that no report date is to appear on the page

If this keyword is omitted, the report date will be placed in the upper-left part of the page.

PAGEPOS position-code
PPOS

Specifies the position on the page at which the page number is to be placed. The valid position codes are as follows:

- UL -- Specifies the upper-left part of the page
- UR -- Specifies the upper-right part of the page
- MT -- Specifies the middle top part of the page
- LL -- Specifies the lower-left part of the page
- LR -- Specifies the lower-right part of the page
- MB -- Specifies the middle bottom part of the page
- NP -- Specifies that no page number is to appear on the page

If this keyword is omitted, the page number will be placed in the upper-right part of the page.

TITLEPOS title-position
TPOS

Specifies the position on the page at which the title lines are to be placed. If this keyword is omitted or title-position is specified as TOP, any title lines will be positioned at the top of the page. If title-position is specified as BOTTOM, any title lines will be positioned at the bottom of the page.

HEADINGS heading-type
HEAD

Specifies whether the report will have a column-heading section and, if so, the source of the heading text. The valid heading-type specifications are as follows:

- YES -- Specifies that this report will contain a column-heading section and that the column text will be the column headings contained in the file definition.
- NO -- Specifies that this report will not contain a column-heading section.
- NAME -- Specifies that this report will contain a column-heading section and the column heading text will be the names of the fields contained in the report.

HEADPOS heading-position
HPOS

Specifies the position on the page at which the column-heading section, if any, is to be placed. If this keyword is omitted or heading-position is specified as ABOVE, the column-heading section will appear above the detail lines. If heading-position is specified as BELOW, the column-heading section will appear below the detail lines.

DATEFMT format-code
DFMT

Specifies a code indicating how the report date is to be formatted or specifies the date to be used as the report date. Valid format codes are as follows:

- DATE -- Specifies that the date be formatted as mmm dd, yyyy (for example, DEC 25, 2001) according to the country conventions specified at installation time in M4PARAMS.
- TODAY -- Specifies that the date be formatted as mm/dd/yy (for example, 12/25/01) according to the country conventions specified at installation time in M4PARAMS.
- TODAYX -- Specifies that the date be formatted as mm/dd/yyyy (for example, 12/25/2001) according to the country conventions specified at installation time in M4PARAMS.
- ISDATE -- Specifies that the date be formatted as yyyy-mm-dd (for example, 2001-12-25).
- JULIAN -- Specifies that the date be formatted as yyddd where ddd is the serial day number of the year (for example, 01359).
- JULANX -- Specifies that the date be formatted as yyyyddd where ddd is the serial day number of the year (for example, 2001359).
- Alternatively, the date that is to be used as the report date may be specified as mmddy according to the country conventions specified at installation time in M4PARAMS (for example, 070402 would be displayed as 07/04/02 using USA conventions).

If the DATEFMT keyword is omitted, the TODAY format will be assumed.

<p>BORDER <i>border-type</i></p>	<p>Specifies whether or not border lines are to appear above and below the column-heading section or specifies the character to be used to form the border lines. Valid border-type specifications are as follows:</p> <ul style="list-style-type: none"> ■ YES -- Specifies that border lines are to appear surrounding the column-heading section lines and that the character specified at installation time in M4PARAMS is to be used to form the lines. ■ NO -- Specifies that no border lines are to appear. <p>Alternatively, a specific border character from which the border line is to be formed may be specified. If a blank is specified as the border character, then the border lines will be blank lines. If the BORDER keyword is omitted, border lines containing the character specified at installation time in M4PARAMS will appear.</p>
<p>STARTPAGE <i>number</i> START</p>	<p>Specifies either the starting page number for the report, or that page numbers are to be reset at a level 1 control break. Specify a number from 1 to 9999 as the starting page number or the word PAGE to specify that page numbers are to be reset at a level 1 control break. If this keyword is omitted, the report page numbers will start at 1.</p>
<p>MAXPAGES <i>number</i> MAXP</p>	<p>Specifies the maximum number of pages that will be produced for the report. When the page count exceeds the maximum number of pages and there is still more data that was selected for the report, the remaining data will be discarded. If this keyword is omitted, all selected data will be output.</p>
<p>LINESPERPAGE <i>number</i> LINES</p>	<p>Specifies that the number of lines printed on each page be limited to the specified number. If this keyword is omitted, the number of lines is determined by the height of the page as specified by the HEIGHT keyword (or its default) less the standard margins.</p>

<p>DETAILSPACING <i>number</i> SPACING</p>	<p>Specifies the spacing between logical detail lines in the report. The number may be in the range of 1 to 9. The number of blank lines that are output between logical detail lines is 1 less than the number specified. For example, if number is 1, no blank lines will be output between logical detail lines; if number is 2, one blank line will be output between logical detail lines.</p>
<p>INCOMPLETESUM 'x' ISUM</p>	<p>Specifies a character that is to print in place of a summary value for summaries that are incomplete due to one or more missing or invalid fields involved in computing the summary. If this keyword is omitted, the computed summary value, ignoring any missing or invalid fields, will be printed.</p>
<p>NODATA action</p>	<p>Specifies the action to be taken when no data is selected for the report. The valid action specifications are:</p> <ul style="list-style-type: none"> ■ SKELETON -- Specifies that a skeleton report is to be produced. ■ NOREPORT -- Specifies that all report output is suppressed. <p>If this keyword is omitted, the default specified action specified at installation time in M4PARAMS will be used.</p>

SUBTITLE action

Specifies the action to be taken when a page break occurs that does not coincide with the subtitle break. See the GROUP command for information regarding subtitles. The valid action specifications are:

- REPEAT -- Specifies that the subtitle line should be repeated before the first detail line is output.
- NOREPEAT -- Specifies that the subtitle line should not be repeated before the first detail line is output.
- NEWPAGE -- Specifies that the repeating subtitles print only on each new physical page of a multi-image report. This is in contrast to the specification of REPEAT (see above) in which case, the repeating subtitles would print on each new logical page. The specification of NEWPAGE is ignored for single-image reports.

SUMMARYLABELS
alignment
LABELS

Specifies how report columns are to be aligned in relation to summary descriptions when summaries are requested. Valid alignment specifications are as follows:

- SPACE -- Specifies that 14 spaces are reserved at the left margin of the page for summary labels. Data columns will begin to the right of the summary labels. This is the default if the SUMMARYLABELS keyword is omitted.
- NOSPACE -- Specifies that no space will be reserved for the summary labels and that data columns will begin at the left margin of the page. Summary labels will appear at the left margin of the page underneath the leftmost data columns.
- SUPPRESS -- Specifies that summary labels are suppressed altogether. Summary values will appear under their respective data columns without any label information.

LINENUMS position
LNUMS

Specifies whether logical detail lines of the report should be numbered and, if so, the position of the line numbers. Valid position specifications are as follows:

- NONE -- Specifies that lines will not be numbered. This is the default if the LINENUMS keyword is omitted.
- LEFT -- Specifies that lines will be numbered and the line numbers will appear at the left margin of the page.
- RIGHT -- Specifies that lines will be numbered and the line numbers will appear at the right margin of the page.
- BOTH -- Specifies that lines will be numbered and the line numbers will appear at both the left and right margins of the page.

If line numbers are specified, 6 positions are reserved for the number. On folded lines, left line numbers print on the first physical line and right line numbers print on the last physical line. Line numbers are reset whenever page numbers are reset.

IMAGES *number*

Specifies the number of logical page images that are to be formatted onto a single physical page. If this keyword is omitted, only 1 logical page will be formatted for a physical page.

IMGTITLE type

Specifies how titles are formatted when the physical page contains more than 1 logical page. Valid type specifications are as follows:

- LOGPAGE -- Specifies that the page title appear at the top of each logical page. This is the default if the IMGTITLE keyword is omitted.
- PHYPAGE -- Specifies that only a single title, spanning the entire physical page, appear at the top of the physical page.
- NEWPAGE -- Specifies that in addition to the layout as with PHYPAGE above, a change in value of page subtitles, if any, cause a skip to the next physical page.

The IMGTITLE keyword is ignored if the IMAGES keyword and a number greater than 1 are not specified.

METHOD output-method

Specifies the output-method to be used for outputting the selected data. Valid output-method specifications are as follows:

- **STDLIST** -- Specifies that the selected data be formatted for a printed page and directed to DD name M4LIST unless the LISTCNTL ALTLIST command is present. In this case, the output will be directed to DD name M4LIST1.
- **ALTLIST** -- Specifies that the selected data be formatted for a printed page and directed to the DD name specified with the DDNAME keyword or to M4LIST1 when the DDNAME keyword is not present.
- **CSV** -- Specifies that the selected data is to be output as comma separated variable data. Column headings will also be formatted one time as the first line(s) of comma separated variable data unless the HEADINGS NO keyword is present. The DDNAME keyword is required when CSV is specified.
- **TAB** -- Specifies that the selected data is to output as tab delimited data. As with the CSV method, column headings will also be formatted one time as the first line(s) of tab delimited output unless the HEADINGS NO keyword is present. The DDNAME keyword is required when TAB is specified.
- **HTML** -- Specifies that the selected data is to be formatted as an HTML document suitable for viewing with a browser. The DDNAME keyword is required when HTML is specified and the dataset associated with the DD name must either be a partitioned dataset (PDS) or a hierarchical file system (HFS) file.

- PLAINTEXT -- Specifies that the selected data be formatted for a printed page as with STDLIST and ALTLIST above, except that the data lines will not contain any printer codes in position 1. Instead, the required line spacing will be achieved by including blank lines as appropriate to create the desired spacing of report lines. The DDNAME keyword is required when PLAINTEXT is specified.
- RAWDATA -- Specifies that the selected data be output as is, in the same manner as with subfile output, except that the data will be sorted per the ORDER command specifications and the output records written during the report phase rather than the processing phase.

STYLE <i>number</i>	Specifies the HTML template to be used in preparing the HTML document from the selected data. The template number may be a value from 1 through 99. If this keyword is omitted, the default template will be used. See the <i>VISION:Builder Environment Guide for OS/390</i> for information regarding the creation of HTML templates.
DDNAME <i>ddname</i>	Specifies the name of the DD to which the output for this report is directed.
SUMFILE <i>ddname</i>	Specifies that a report summary file is to be created and the DD name to which the file is to be directed. A report summary file is a data file that contains the summary data records in a format that is suitable for post-processing.
AUTODEF	Specifies that file definition statements be generated for the data elements contained in this report when METHOD RAWDATA is specified or that file definition statements be generated for the report summary data file when SUMFILE is specified. A FILE SUBFn command with the AUTODEF keyword must be present in the Run Control section of the application when the AUTODEF keyword is specified on the FORMAT statement.

Examples

```
FORMAT WIDTH 80, HEADINGS NAME,
```

```
FORMAT METHOD HTML, STYLE 5, DDNAME HTMLOUT1
```

```
FORMAT BORDER '.', SUBTITLE REPEAT, IMAGES 2, IMGTITLE PHYPAGE
```

GROUP Command

```
GROUP [BY] field-name ...,  
      [AT LEVEL level-number],  
      [SUBTITLE [NEWPAGE]],  
      [LABEL field-name]
```

The GROUP command specifies how the selected report data is to be grouped. Additionally, it may be used to specify that the grouping field is to be placed in a line called a subtitle line rather than in a detail line.

<i>BY field-name ...</i>	Specifies the qualified names of one or more fields that will be used to determine the control break for this group. The field may be a partial field using the PF function notation.
AT LEVEL <i>level-number</i>	Specifies that this set of grouping fields should be considered the grouping fields for the indicated level number. Valid specifications for number are 1 through 9. If this keyword is omitted, the level number will be incremented by 1 from the level number on the previous GROUP command. If this keyword is omitted from the very first GROUP command, the first level number will be assumed to be 1 and will be incremented for each successive GROUP command without an AT LEVEL keyword.
SUBTITLE	Specifies that the fields identified with the BY keyword are to be displayed as subtitle fields. This means that the fields are not placed into the detail report line but will appear on a line preceding the detail lines as a group heading for this group. These group headings will appear within the detail section of the report page.
NEWPAGE	Specifies that when a control break occurs for the fields in this group, a new page should be formatted. Furthermore, this keyword specifies that the group heading line will appear in the title section of the page after the page title lines but before the column heading lines.
LABEL <i>field-name</i>	Specifies the qualified name of the field whose value will be used in the summary label at the corresponding level when summaries are specified for the same level or BY keyword fields as this GROUP statement. The field may be a partial field using the PF function notation.

Examples

GROUP BY DEPT, SUBTITLE NEWPAGE

GROUP BY ZIPCODE AT LEVEL 3

ITEM Command

```
ITEM [COLUMN] field-name ...,  
    [SPACES number],  
    [PICTURE P'pattern' ],  
    [ENDLINE],  
    [NONPRINT],  
    [VWIDTH],  
    [NOWRAP],  
    [SPLITOK],  
    [CSVEDIT {QUOTE | TRUNCATE [DECIMALS number]}]
```

The ITEM command is used to specify special controls for columns specified on the REPORT command with the COLUMNS keyword.

COLUMN <i>field-name</i> ... COLUMNS	Specifies the qualified names of one or more columns included in the report to which the controls on this ITEM statement apply. The field may be a partial field using the PF function notation. The qualified field names, including partial field specifications if any, must match one of the fields specified on the REPORT command ITEMS keyword list. If the qualified field name, including partial field specifications if any, matches more than one field in the REPORT ITEMS list, the ITEM command specifications will only apply to the first occurrence of that field.
SPACES <i>number</i>	Specifies the number of spaces that are to precede the column for this field.
PICTURE P' <i>pattern</i> ' PIC	Specifies the edit picture that defines how the data in this field is to be presented. The picture representation depends upon the type of field -- character fields, date fields, or numeric fields. See Appendix B for valid edit pattern symbols for each type of field.
ENDLINE	Specifies that this field is the last (rightmost) field for this physical line and that the next field, if any, begins a new physical line.
NONPRINT NOPRINT	Specifies that the field is selected for control purposes only and should not be included in any output for this report.

VWIDTH <i>number</i>	Specifies the column width for the associated V-type field. Data from the V-type field will be folded into multiple physical lines within the specified column width. Valid specifications for number are 1 up the report page width (see WIDTH keyword in FORMAT command). If this keyword is omitted, the column width will be assumed as the lesser of the remaining positions on the line up to the right margin or the length of the V-type field.
NOWRAP	Specifies that word wrapping should not be used when formatting this V-type field. Instead, the exact number of characters assigned for the column width (see keyword VWIDTH above) should be placed into each physical line containing data for this field without regard to word boundaries. If this keyword is omitted, the V-type field data will be folded at word boundaries unless the length of the word exceeds the width of the column.
SPLITOK	Specifies that formatting for this V-type field should begin at the current vertical page position and then be continued on the next page if the bottom of page is reached before the entire V-type field is formatted. If this keyword is omitted and the entire V-type field will not fit onto the remainder of the current page, formatting of the V-type field will begin on a new page.
CSVEDIT action CSV	Specifies that the specified action is to be taken when the FORMAT command METHOD CSV keyword is present. Valid specifications for action are as follows: <ul style="list-style-type: none">■ QUOTE -- Specifies that the field is to be enclosed in quotes (") regardless of whether the field contains an embedded comma. CSVEDIT QUOTE may only be specified for character type or V-type fields. If CSVEDIT QUOTE is not specified, the field will only be quoted when it contains an embedded comma.■ TRUNCATE -- Specifies that non-significant trailing zeros are to be truncated up to the limit specified with the DECIMALS keyword. CSVEDIT TRUNCATE may only be specified for numeric type fields. If the DECIMALS keyword is omitted but CSVEDIT TRUNCATE is specified, all non-significant trailing zeros are truncated.

DECIMALS *number*
DEC

Specifies the number of positions to the right of the decimal point that should be preserved during a CSVEDIT TRUNCATE process even though they may contain non-significant zeros. The valid specification for number is 0 through 9.

Examples

```
ITEM T.TEMP1, NONPRINT  
ITEM DESCRIPT, VWIDTH 40, SPLITOK  
ITEM ZIPCODE, CSVEDIT QUOTE
```

LINE Command

```
LINE [{ 'text' | LIT[ERAL]('text', repeat-count) | field-name |,
      summary-function(field-name [level]) |,
      STEMPnn | COL(col-num) | SPACES(num-spaces)}...] [AT LEVEL level]
```

The LINE command is used to output a line for the corresponding section of a sectional report (see the SECTION command). For PAGETITLE or COLUMNHEADING sections, the content and layout of the report line must be specified on the LINE command itself. For a SUMMARY section, the content and layout of a line may be specified by optional DATA commands followed by a LINE command.

'text'	Specifies literal text that is to be placed into the report line at the current position.
LITERAL('text', repeat-count)	Specifies literal text that is to be placed into the report line at the current position and repeated repeat-count times.
LIT	
field-name	Specifies the name of a field whose contents are to be placed into the report line at the current position.

summary-function(*field-name* [*level*])

Specifies that a summary value for the field identified by *field-name* whose summary type specified by *summary-function* is to be placed into the report line at the current position. *Summary-function* may be specified as any of the TOTAL[TOT], CUMULATE[CUM], COUNT[CNT], MAX, MIN, AVERAGE[AVG], PERCENT[PCT], or RATIO[RTO] summaries. When *summary-function* is specified, a corresponding summary command (for example, TOTAL command) or a REPORT command ITEMS operand summary function (for example, TOTAL(*field-name* BY *field-name*)) must also be specified. The *summary-function level number operand* specifies the control break level of the summary to be used and is optional. If specified, it must be a number between 1 and 9 or G and must be at a level equal to or lower (numerically higher) than the level number specified by AT LEVEL. If the *summary-function level number* is not specified, it will default to the level number specified with the AT LEVEL keyword. The *summary-function operand* may only be used when the LINE command is part of a summary section. Partial field notation (PF function) may not be used for the field.

Summary-function operands are only permitted if the LINE command is within a SUMMARY section.

STEMPnn

Specifies the STEM P field that is to be placed into the report line at the current position. The STEM P operand may only be used when the LINE command is part of a summary section. STEM P fields must have been defined with a COMPUTE command prior to the LINE command. Refer to the COMPUTE command for further detail regarding the definition and use of STEM P fields.

STEM P fields are only permitted if the LINE command is within a SUMMARY section.

<code>COL(<i>column-number</i>)</code>	Specifies that the current columnar position within the print line is to be set to <i>column-number</i> .
<code>SPACES(<i>number-spaces</i>)</code>	Specifies that the current columnar position within the print line is to be advanced <i>number-spaces</i> positions.
<code>AT LEVEL <i>level-number</i></code>	Specifies the control break level number at which the respective data elements are to be placed into the report line. Valid specifications for level number are 1 through 9 and G. The report line is output when the control break level is equal to or higher (numerically lower) than the level number. This keyword is only permitted if the LINE command is within a summary section. If this keyword is omitted, the line will always be output.

Examples

```
DATA COL(60) 'DEPARTMENT TOTAL: ' TOTAL(AMOUNT) AT LEVEL 2
DATA COL(30) 'DIVISION TOTAL: '   TOTAL(AMOUNT) AT LEVEL 1
DATA COL(1)  'GRAND TOTAL: '      TOTAL(AMOUNT) AT LEVEL G
LINE
```

Output a line containing the data from the prior DATA commands.

```
LINE COL(35) 'INVENTORY STATUS REPORT'
```

Output a line containing the specified data.

MAX Command

```
MAX [ITEM] summary-field ...,  
      {BY group-field | AT LEVEL level-number}
```

The MAX command is used to specify that a maximum value type summary is to be determined for the field.

ITEM <i>summary-field</i> ...	Specifies the qualified names of one or more fields for which a maximum value is to be determined. The field may be a partial field using the PF function notation.
BY <i>group-field</i>	Specifies the qualified name of the field by which the summaries are to be grouped. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the AT LEVEL keyword.
AT LEVEL <i>level-number</i>	Specifies a control break level number by which the summaries are to be grouped. Valid specifications for level number are 1 through 9 and G. If this keyword is specified, a GROUP command with an AT LEVEL keyword must also be specified. The GROUP command specifies the name(s) of the field(s) by which the summaries are grouped. This keyword is mutually exclusive with the BY keyword.

Examples

```
MAX AMOUNT BY CUSTNO
```

Determine the maximum value of the field named AMOUNT grouped by CUSTNO.

```
MAX SALARY AT LEVEL 2
```

Determine the maximum of the field named SALARY at the level 2 control break.

MIN Command

```
MIN [ITEM] summary-field ...,
      {BY group-field | AT LEVEL level-number}
```

The MIN command is used to specify that a minimum value type summary is to be determined for the field.

<i>ITEM summary-field ...</i>	Specifies the qualified names of one or more fields for which a minimum value is to be determined. The field may be a partial field using the PF function notation.
<i>BY group-field</i>	Specifies the qualified name of the field by which the summaries are to be grouped. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the AT LEVEL keyword.
<i>AT LEVEL level-number</i>	Specifies a control break level number by which the summaries are to be grouped. Valid specifications for level number are 1 through 9 and G. If this keyword is specified, a GROUP command with an AT LEVEL keyword must also be specified. The GROUP command specifies the name(s) of the field(s) by which the summaries are grouped. This keyword is mutually exclusive with the BY keyword.

Examples

```
MIN AMOUNT BY CUSTNO
```

Determine the minimum value of the field named AMOUNT grouped by CUSTNO.

```
MAX SALARY AT LEVEL 2
```

Determine the minimum of the field named SALARY at the level 2 control break.

NEWPAGE Command

NEWPAGE [**AT LEVEL**] *level-number*

The NEWPAGE command is used to specify the control break level that will force a new page in a formatted report. This command must appear inside of either a DETAIL section or a SUMMARY section (see SECTION command). If the formatted report includes a DETAIL section, the NEWPAGE command must appear there and the report must not be a summary only report. If the report is a summary only report, the NEWPAGE command must appear inside the SUMMARY section.

AT LEVEL *level-number* Specifies a control break level that will force a new page to be output. When the NEWPAGE command is inside a DETAIL section, the level number may be 1 through 9 or G. When the NEWPAGE command is inside a SUMMARY section, the level number may be 0 through 9. A value of 0 in this case means that a page break only occurs when the current page has been filled (no space left to print another set of summaries).

ORDER Command

```
ORDER [BY] field-name ...,  
      [DESC field-name ...]
```

The ORDER command is used to specify the order in which the selected data records are to be sorted. Multiple ORDER commands may be provided. The first field specified on the first ORDER command is the most major sort field, the second field on the first command the next major, and so on. The fields on the second and subsequent ORDER commands become increasingly minor. A maximum of 9 fields may be specified on all GROUP commands.

BY <i>field-name</i> ...	Specifies the qualified names of one or more fields by which the selected data is to be ordered. The field may be a partial field using the PF function notation. The ordering will be in ascending order unless the DESC keyword (see below) is used as an override.
DESC <i>field-name</i> ...	Specifies the qualified names of one or more fields, including partial field specifications if any, from the list of fields specified with the BY keyword (see above) that are to be ordered in descending order rather than ascending order. The field may be a partial field using the PF function notation.

Examples

```
ORDER BY ZIPCODE PF(NAME 1 3), DESC PF(NAME 1 3)
```

Order the report data first in ascending order by ZIPCODE (major key) and then in descending order by the first 3 characters of NAME (minor key).

```
ORDER BY CUSTNO PRODTYPE DATE
```

Order the report data first in ascending order by CUSTNO (major key) and then by PRODTYPE (minor key) and DATE (next minor key).

PERCENT Command

```
PERCENT [ITEM] summary-field ... OF denominator-field,  
      {BY group-field | AT LEVEL level-number},  
      [DECIMALS decimal-places]
```

The PERCENT command is used to specify that a percent value type summary is to be determined for the field. The percent is computed by dividing the ITEM field value by the OF field value and multiplying by 100. PCT may be used as an abbreviation for PERCENT.

ITEM <i>summary-field</i> ...	Specifies the qualified names of one or more fields for which a percent value is to be determined. The field may be a partial field using the PF function notation.
OF <i>denominator-field</i>	Specifies the qualified name of the field that will be used as the denominator for computing the percent value.
BY <i>group-field</i>	Specifies the qualified name of the field by which the summaries are to be grouped. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the AT LEVEL keyword.
AT LEVEL <i>level-number</i>	Specifies a control break level number by which the summaries are to be grouped. Valid specifications for level number are 1 through 9 and G. If this keyword is specified, a GROUP command with an AT LEVEL keyword must also be specified. The GROUP command specifies the name(s) of the field(s) by which the summaries are grouped. This keyword is mutually exclusive with the BY keyword.
DECIMALS <i>decimal-places</i>	Specifies the number of decimal places that are to be preserved in the result of the percent computation. If this keyword is omitted, 2 decimal places will be preserved.

Examples

```
PERCENT NET_AMOUNT OF GROSS_AMOUNT BY STATE
```

Compute the percent that NET_AMOUNT is of GROSS_AMOUNT grouped by STATE.

```
PERCENT ACTUAL OF ESTIMATED AT LEVEL 2 DECIMALS 3
```

Compute the percent that ACTUAL is of ESTIMATED to 3 decimal places at the level 2 control break.

PREFACE Command

PREFACE [**LINE**] '*text*' ...

The PREFACE command is used to specify text that is part of the report preface page. Each PREFACE command specifies one line of the preface page, and the PREFACE command is repeated as many times as needed. If no PREFACE commands are provided, no report preface page will be produced.

LINE '*text*' ... Specifies one or more segments of text for a preface line. Concatenating all of the text segments together will form the complete preface line. The concatenated length may not exceed the page width. The text operands can be enclosed in double quotation marks (") in place of the single quotation marks (').

Example

```
PREFACE 'This report is a listing of outstanding ',  
        'invoices by customer.'  
PREFACE 'Customers are listed alphabetically by name and ',  
        'invoices are listed in descending order by due date.'
```

This example would produce a report preface containing 2 lines of text.

RATIO Command

```
RATIO [ITEM] summary-field ..., TO denominator-field,
      {BY group-field | AT LEVEL level-number},
      [DECIMALS decimal-places]
```

The RATIO command is used to specify that a ratio value type summary is to be determined for the field. The ratio is computed by dividing the ITEM field value by the TO field value. RTO may be used as an abbreviation for RATIO.

ITEM <i>summary-field</i> ...	Specifies the qualified names of one or more fields for which a ratio value is to be determined. The field may be a partial field using the PF function notation.
TO <i>denominator-field</i>	Specifies the qualified name of the field that will be used as the denominator for computing the ratio value.
BY <i>group-field</i>	Specifies the qualified name of the field by which the summaries are to be grouped. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the AT LEVEL keyword.
AT LEVEL <i>level-number</i>	Specifies a control break level number by which the summaries are to be grouped. Valid specifications for level number are 1 through 9 and G. If this keyword is specified, a GROUP command with an AT LEVEL keyword must also be specified. The GROUP command specifies the name(s) of the field(s) by which the summaries are grouped. This keyword is mutually exclusive with the BY keyword.
DECIMALS <i>decimal-places</i>	Specifies the number of decimal places that are to be preserved in the result of the ratio computation. If this keyword is omitted, 3 decimal places will be preserved.

Examples

```
RATIO NET_AMOUNT TO GROSS_AMOUNT BY STATE
```

Compute the ratio of NET_AMOUNT to GROSS_AMOUNT grouped by STATE.

```
RATIO ACTUAL TO ESTIMATED AT LEVEL 2 DECIMALS 3
```

Compute the ratio of ACTUAL to ESTIMATED to 3 decimal places at the level 2 control break.

REPORT Command

```
[report-name:]REPORT [COLUMNS] {field-name | report-function} ...,
    [SUMMARYONLY],
    [SINGLESPACE],
    [GRANDSUMS],
    [EMPTYFIELD {INCLUDE | EXCLUDE}],
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [ABBREVIATED],
    [FILE report-file-name],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE_MASTER_READ |
          TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
          EOF | EOFPLUS}],
    [INFO 'text']
```

The REPORT command begins a report block and specifies report content as well as several other parameters and controls. A report block, headed by a REPORT command, may appear either as a self-contained element within an application or embedded inside a procedure block.

Within the context of VISION:Builder, a report is a logical entity that does not necessarily mean that a printed report is produced. The term report is used to represent selected data output in a variety of formats for printing, viewing, or processing by other applications. Additional report parameters and controls may be specified using the FORMAT command. See the FORMAT command METHOD keyword for information regarding the various ways in which the selected report data can be output.

report-name:	Specifies the name of the report. Naming a report is optional and is only required if the report is referenced in a Run Control COLLATE or ROUTE statement or the report type is declared as SUBROUTINE.
COLUMNS <i>field-name ...</i> COLUMN	Specifies the content of the report as a list of fields that will be included in the report. The fields may be partial fields using the PF function notation or built-in report functions. Each field in the operand list will represent one column in the report, and the columns will be ordered in the same order as the items in the list. The ITEM command (see above) may be used to specify additional properties for the fields represented by the operands in this list.

The following list describes the syntax of the PF function and the available built-in report functions that may be used as ITEM operands:

- PF([FIELD] field-name [START] start-position [LENGTH] partial-length)
 - This function is used to specify that the item is a partial field.
- TOTAL([ITEM] summary-field [BY] group-field)
 - This function is used to specify a report column and, in addition, that a total summary is to be computed for the column. The TOTAL function may be specified in place of a TOTAL command. Partial field notation (PF function) may be used for both summary-field and group-field. TOT may be used as an abbreviation for TOTAL.
- CUMULATE([ITEM] summary-field [BY] group-field)
 - This function is used to specify a report column and, in addition, that a cumulative total summary is to be computed for the column. The CUMULATE function may be specified in place of a CUMULATE command. Partial field notation (PF function) may be used for both summary-field and group-field. CUM may be used as an abbreviation for CUMULATE.

- COUNT([ITEM] summary-field [BY] group-field)
 - This function is used to specify a report column and, in addition, that a count summary is to be computed for the column. The COUNT function may be specified in place of a COUNT command. Partial field notation (PF function) may be used for both summary-field and group-field. CNT may be used as an abbreviation for COUNT.
- MAX([ITEM] summary-field [BY] group-field)
 - This function is used to specify a report column and, in addition, that a maximum value summary is to be computed for the column. The MAX function may be specified in place of a MAX command. Partial field notation (PF function) may be used for both summary-field and group-field.
- MIN([ITEM] summary-field [BY] group-field)
 - This function is used to specify a report column and, in addition, that a minimum value summary is to be computed for the column. The MIN function may be specified in place of a MIN command. Partial field notation (PF function) may be used for both summary-field and group-field.
- AVERAGE([ITEM] summary-field [BY] group-field)
 - This function is used to specify a report column and, in addition, that an average value summary is to be computed for the column. The AVG function may be specified in place of an AVG command. Partial field notation (PF function) may be used for both summary-field and group-field. AVG may be used as an abbreviation for AVERAGE.

SUMMARYONLY SUMMARY	Specifies that the report is to be a summary only report. The report will not contain any detail data. If this keyword is omitted, the report will contain both detail and summary lines when summaries are specified.
SINGLES SPACE SINGLE	Specifies that all summary lines are to be single-spaced except that summary lines for the level 1 group will be double-spaced. Beginning the GROUP levels at level number 2 rather than level 1 (the default) is a way to cause all summary lines to be single-spaced. This keyword may only be specified if the SUMMARYONLY keyword is also specified. If this keyword is omitted when the SUMMARYONLY keyword is specified, all summary lines will be double-spaced.
GRANDSUMS GRAND	Specifies that grand summaries are to be automatically computed for every summary specified for this report. If this keyword is omitted, the default specified at installation time in M4PARAMS will be used to determine if automatic grand summaries are to be computed.
EMPTYFIELD {INCLUDE EXCLUDE} EMPTY	<p>Specifies the action to be taken for empty fields when a summary is being computed. An empty field is a field that is either blank (character string type) or zero (numeric type).</p> <ul style="list-style-type: none">■ INCLUDE -- Specifies that an empty field is to be included in the computation of the summary for this field.■ EXCLUDE -- Specifies that an empty field is to be excluded in the computation of the summary for this field. <p>If this keyword is omitted, empty numeric fields (contains zero) are included in the computation of the summary, while empty character string fields (contains blanks) are excluded in the computation of the summary.</p>

SELECTIONCONTROL {YES | NO}
SELCNTL

Specifies whether or not all occurrences of repeated segments in a hierarchical structure are to be examined.

- YES -- Specifies that only the first occurrence of a repeated segment within each record that meets the selection qualification will be selected.
- NO -- Specifies that all occurrences of repeated segments within each record that meet the selection qualification will be selected. This is the default if the SELECTIONCONTROL keyword is omitted.

MAXITEMS *number*

Specifies the maximum number of records that will be selected for this report. Specify a number from 1 to 9999 to limit the number of records that will be selected for this report. If this keyword is omitted, the number of records that will be selected is unlimited.

ABBREVIATED
ABBREV

Specifies whether repeated selections of the same data will be output only once. This use of this keyword is only meaningful when one or more of the input files contain data in a hierarchical structure wherein the segments have multiple branches or when more than one standard coordinated file (see the FILE CORDn command) is used. These coordinated files may be viewed as a hierarchical structure with multiple branches.

If this keyword is specified, an output record will only be produced the first time that each unique combination of different repeating segments is examined and the qualifying criteria are met. If this keyword is omitted, an output record will be produced each time any combination of different repeating segments is examined and the qualifying criteria are met.

Note: In earlier versions of VISION:Builder, this functionality was achieved through a specification called Parallel Looping. The use of ABBREVIATED here limits the action of Parallel Looping to output operations only.

FILE *report-file-name*

Specifies the name of the report file to be used for this report (see the Run Control FILE REPORT command). If this keyword is omitted, the report data will be output to the default report file, M4REPO.

TYPE invocation-event

Specifies the event within the application processing cycle at which this report output will be invoked. This keyword is invalid if the report block is embedded inside a procedure. If this keyword is omitted, the report output will be invoked during each iteration of the master file read and standard coordination cycles. Valid invocation-event specifications are:

- **NORMAL** -- Specifies that the report output will be invoked during each iteration of the master file read and standard coordination cycles.
- **SUBROUTINE** or **SUB** -- Specifies that the report output will only be invoked when explicitly called by a procedural **CALL** statement.
- **INIT** -- Specifies that the report output will be invoked only one time after the master and coordinated files have been opened but before any records from these files have been processed.
- **PRE_MASTER_READ** -- Specifies that the report output will be invoked just prior to reading the next master file record.
- **TYPE1** -- Specifies that the report output will be invoked after each transaction record has been read.
- **TYPE2** -- Specifies that the report output will be invoked after the transaction file and the master files have been aligned.
- **TYPEN** -- Specifies that the report output will be invoked after the master file record has been updated but before first round coordination.
- **TYPE3** -- Specifies that the report output will be invoked after the master file record has been updated but after first round coordination.

- TYPE4 -- Specifies that the report output will be invoked whenever a transaction record is rejected, either explicitly by procedural code or implicitly by transaction validation controls.
- EOF -- Specifies that the report output will be invoked only once after the transaction file, master file, and all coordinated files have reached end of file.
- EOFPLUS -- Specifies that the report output will be invoked as in NORMAL and additionally as in EOF above.

INFO 'text'

Specifies descriptive text that may be used to annotate the report. The text cannot be longer than 28 characters.

Examples

```
PROC
  IF HIREDATE - BIRTHDATE LT 30*365 ;Hired at age 30 or less
    REPORT NAME, ADDRESS, PHONE
      FORMAT DATEPOS NO, PAGEPOS NO, HEADINGS NO,
        METHOD PLAINTEXT, DDNAME LIST1
    END REPORT
  END IF
END PROC
```

The above example shows a report block embedded within a procedure block.

```
REPORT DEPTNAME TOTAL(SALARY BY DEPTNAME), SUMMARYONLY
  ORDER BY DEPTNAME
  GROUP BY DEPTNAME
END REPORT
```

The above example shows a simple summary only report listing DEPTNAME and SALARY totals by DEPTNAME.

```
PROC
  CALL REPORT REP1
END PROC
;
REP1: REPORT PF(ZIPCODE 1 5) NAME ADDRESS, TYPE SUBROUTINE
  ORDER BY ZIPCODE
  GROUP BY ZIPCODE
  COUNT NAME BY ZIPCODE
END REPORT
```

The above example shows a report defined as a subroutine being called from a procedure.

SECTION Command

SECTION {PAGETITLE | COLUMNHEADING | DETAIL | SUMMARY}

The SECTION command is used to specify the beginning of a section block that defines the content and layout for the section of a formatted report.

PAGETITLE	Specifies that the following LINE and SKIP commands are for the page title section of the report.
COLUMNHEADING	Specifies that the following LINE and SKIP commands are for the column heading section of the report.
DETAIL	Specifies that the report is a formatted report. The following required SIZE command specifies the portion of the page reserved for the detail section. The optional NEWPAGE command may be used to specify the controls for beginning a new page of the report.
SUMMARY	Specifies that the following COMPUTE, DATA, LINE, NEWPAGE, and SKIP commands are for the summary section of the report.

Examples

```
SECTION PAGETITLE
  LINE COL(35) 'Weekly Status Report - Week Ending ' F.DATE
END SECTION
```

The above example shows a simple page title section.

```
REPORT GROUPNO DEPTNO EMPNO SALARY
ORDER BY GROUPNO DEPTNO
GROUP BY GROUPNO AT LEVEL 1
GROUP BY DEPTNO AT LEVEL 2
SECTION SUMMARY
  LINE 'Department Totals ' TOTAL(AMOUNT) AT LEVEL 2
  LINE 'Group Totals      ' TOTAL(AMOUNT) AT LEVEL 1
END SECTION
END REPORT
```

The above example shows a section block as it would appear inside of a report block.

SIZE Command

SIZE *number-of-lines* [LINES]

The SIZE command is used to specify the number of lines on the page to reserve for the detail section of the formatted report.

number-of-lines LINES Specifies the number of lines to be reserved for the detail section.

SKIP Command

SKIP *number-of-lines* LINES [AT LEVEL *level-number*]

The SKIP command is used to specify that blank lines are to be output for the section.

<i>number-of-lines</i> LINES LINE	Specifies the number of blank lines that are to be output.
AT LEVEL <i>level-number</i>	Specifies that the blank lines are to be output only when a control break at a level equal to or higher (numerically lower) than <i>level-number</i> occurs. Level number may be specified as 1 through 9 or G. The AT LEVEL keyword may only be specified when the SKIP command is part of a summary section.

Examples

```
SKIP 3 LINES
```

The above example would output 3 blank lines in the section.

```
REPORT STATE ZIPCODE NAME
ORDER BY STATE ZIPCODE NAME
GROUP BY STATE
GROUP BY ZIPCODE
COUNT NAME BY ZIPCODE
SECTION SUMMARY
  SKIP 1 LINE AT LEVEL 2
  LINE 'Count for ' ZIPCODE SPACES (2) COUNT (NAME) AT LEVEL 2
  SKIP 2 LINES AT LEVEL 1
  LINE 'Count for ' STATE SPACES (2) COUNT (NAME) AT LEVEL 1
END SECTION
END REPORT
```

The above example would output 1 blank line followed by the ZIPCODE summary line at the level 2 control break, and 2 blank lines followed by the STATE summary line at the level 1 control break.

TITLE Command

TITLE [**LINE**] '*text*' ...

The TITLE command is used to specify text that is part of the title section of the report. Each TITLE command specifies one line of the title section, and the TITLE command is repeated as many times as needed. If no TITLE commands are provided, the title section will consist of one line containing the report date and page number unless these are also suppressed through the FORMAT command DATEPOS NO and PAGEPOS NO keywords.

LINE '*text*' ... Specifies one or more segments of text for a title line. Concatenating all of the text segments together will form the complete title line. The concatenated length may not exceed the page width. The title lines will be centered on the page. Leading or trailing blanks can be imbedded in the text to offset the centering, if so desired. The text operands can be enclosed in double quotation marks (") in place of the single quotation marks (').

Example

```
TITLE 'ACCOUNTS RECEIVABLE'  
TITLE 'AGING REPORT'
```

This example would produce a title section containing 2 lines of text.

TOTAL Command

```
TOTAL [ITEM] summary-field ...,  
      {BY group-field | AT LEVEL level-number}
```

The TOTAL command is used to specify that a total type summary is to be computed for the field.

ITEM summary-field ...	Specifies the qualified names of one or more fields for which a total is to be computed. The field may be a partial field using the PF function notation.
BY <i>group-field</i>	Specifies the qualified name of the field by which the summaries are to be grouped. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the AT LEVEL keyword.
AT LEVEL <i>level-number</i>	Specifies a control break level number by which the summaries are to be grouped. Valid specifications for level number are 1 through 9 and G. If this keyword is specified, a GROUP command with an AT LEVEL keyword must also be specified. The GROUP command specifies the name(s) of the field(s) by which the summaries are grouped. This keyword is mutually exclusive with the BY keyword.

Examples

```
TOTAL INVAMT BY CUSTNO
```

Compute the total for INVAMT grouped by CUSTNO.

```
TOTAL SALARY AT LEVEL 2
```

Compute the total for SALARY at the level 2 control break.

XREP Command

XREP [**LINE**] '*text*'

The XREP command is used to specify dynamic report line modification controls for the report. Dynamic report line modification controls are applied to formatted report lines just prior to their output to the report destination. Each XREP command may only specify one control. XREP commands may only be specified when the FORMAT command METHOD keyword specifies STDLIST or ALTLIST.

LINE '*text*' Specifies a dynamic report line modification control. The controls consist of keywords, positional indicators (nnn), operators (=,1), and operands. Because these operations take place after the report line is formatted, generate a sample report first to determine values for positional indicators and operands where needed. The controls may be specified in any order; however, execution is in the order of the commands as listed below. The text operands can be enclosed in double quotation marks (") in place of the single quotation marks ('). Following is a description of each control in the order that they will be executed.

TITLE LINENO=nn

Exempts a specified number of lines (nn = any 2-digit number) from processing by other dynamic report line modification statements.

On an output printer, this specification takes effect whenever a skip to new page (channel 1 punch) occurs.

The TITLE LINENO=nn keyword allows you to modify line spacing or perform calculations without affecting title lines and column headings.

SKIP c POSnnn̄x

Invokes ASA carriage control characters on specific report lines. This is useful when using preprinted forms.

c -- Carriage control channel to be invoked (2 to 9) or print without spacing (+).

nnn -- Any 3-digit number between 001 and 132 that indicates the position where the x character is or is not found.

x -- Character (any printing or nonprinting character) to cause the invoking of c to take place, if:

= x character is found.

* x character is not found.

Note: Up to three SKIP statements are allowed per report. Multiple fonts per printed line are allowed for page printers (laser printers).

NOSPACELINES

Suppresses output of any blank line(s). When lines are suppressed, the line count is decremented; pages are filled as if the blank lines had never been output to the report file. This specification does not affect title lines or those inserted by the SPACEBEFORE or SPACEAFTER keywords.

NOCALC POSnnn \bar{x}

Allows lines meeting specific logical conditions to be exempted from some calculations and formatting (PERCENT, RATIO, ROUND, DECIMALEEDIT, SUPPRESS SPACE).

nnn -- Any 3-digit number between 001 and 132 to indicate the position where the x character is or is not found.

x -- Character (any printing or nonprinting character) to cause the operation of percent ratio, round, decimal edit, or suppress space to be bypassed, if:

= x character is found.

* x character is not found.

Note: The NOCALC keyword can be used to avoid editing or calculating on subtitles or to allow performing some operations on detail lines but not on summaries.

Up to three NOCALC statements are allowed per report.

PERCENT, RATIO, ROUND

Specifies arithmetic operations that allow division or rounding based on data in the output line. These three operations are executed in the processing phase, in order of input, after NOCALC, and before DECIMALEEDIT.

PERCENT A=(nnn,pppp)
B=(nnn,pppp) C=(nnn,pppp)

Calculates $\frac{A}{B} \times 100 = C$. The picture used for each field must correspond exactly to the output. The operand A and B pictures are determined by the VISION:Builder output; the operand C picture is determined by you.

nnn -- Location of first character associated with the field involved in the calculation (001–132).

pppp -- Picture edit in the following ranges:

Operand A and B:

1–11 digits before the decimal point.

Up to 4 digits following the decimal point.

Include any leading or trailing minus signs, as well as commas and decimal points required by the data.

Operand C (Result field):

1–11 digits before the decimal point.

Up to 2 digits following the decimal point.

Commas are not allowed, and a leading minus sign always prints for negative values.

- The picture scan ends when the first nonnumeric character is encountered.
- A trailing % is required in the picture.

- Starting locations of two different operands (nnn) can be the same to allow you to replace a standard output field with one that has been calculated.
- Division by 0 or a field that does not agree with its picture results in a blank output field rather than the invalid field indicator (*).
- Result values that do not fit in the edit picture for operand C produce a plus sign (+) on the output.

RATIO A=(nnn,pppp)
B=(nnn,pppp) C=(nnn,pppp)

Calculates $\frac{A}{B} = C$. The calculation is simple division and can be used as such or as a true ratio.

nnn -- Location of first character associated with the calculation (001–132).

pppp -- Picture edit in the following ranges:

Operand A and B:

A maximum of 4 digits following the decimal point.

Include any leading or trailing minus signs, as well as commas and decimal points required by the data.

Operand C (Result field):

1–9 digits before the decimal point.

At least 1 digit after the decimal point, up to a maximum of 4.

Commas are not allowed, and a leading minus sign always prints for negative values.

Note: In all result fields (operand C) calculated by the dynamic report line modification capability, leading zeroes are suppressed up to the position before the decimal point.

Starting locations of two different operands (nnn) can be the same to allow you to replace a standard output field with one that has been calculated.

ROUND A=(nnn,pppp)

Provides rounding (half adjusting) and formatting.

nnn -- Location of first character associated with the calculation made on it (001–132).

pppp -- Picture edit in the following ranges:

Operand A (Result field):

1–11 digits before the decimal point.

Up to a maximum of 4 digits following the decimal point.

Leading or trailing minus signs can be substituted for one of the digits.

Rounding is done on the field specified in nnn. If the number is positive, 5 is added to the rightmost digit; if negative, 5 is subtracted. In both cases, the last digit of the result is truncated, including the decimal point if there is only one place in the original data field.

Formatting is a result of the picture characteristics of the operand. With this function:

- A sign can be moved from leading to trailing.
- Extra unit positions can be inserted.
- A number between +1 and -1 will be shown with a zero before the decimal point.
- A zero value will print as blanks unless it becomes zero only after the rounding function is completed.

Note: Commas are permitted in the picture and appear in the result when printed.

If the result of the rounding would overflow the size of the picture, the maximum possible value is inserted.

DECIMALEEDIT

Provides extra formatting features for fields with a decimal point, no leading minus sign, and values between +1 and -1. The results are similar to those described for the ROUND keyword.

When the DECIMALEEDIT keyword is used, data values on all output lines not exempted by the TITLE LINENO or NOCALC keywords are modified when they fall in the specified value range:

- On all zero results, printing is suppressed and a blank is inserted in the output.
- At the same time, on all fields with values less than 1 with at least one place after the decimal point and no leading minus sign, a zero is printed before the decimal point.

These two operations occur simultaneously and cannot be invoked separately. When it is necessary to separate the functions, use the ROUND keyword.

The location of the fields is determined by a scan of the first line on the report not controlled by the TITLE LINENO or NOCALC keywords. This determines the position of all decimal points.

- If a column exists without a decimal point on that first line, DECIMALEEDIT will not affect that column on any line.
- Similarly, a period in a character string of the first line will be interpreted as a decimal point and DECIMALEEDIT will be attempted at that column on all nonexempt lines.

- `SUPPRESS SPACE=(nnn,nnn)` Suppresses printing of lines having spaces in specific positions. The parameters `nnn,nnn` specify the starting and ending positions to be examined. When the print image between and including those positions is blank, the report line is suppressed.
- The internal line count is adjusted taking into account the effect of the ASA carriage control character that can appear in position zero (0) of the line. If the carriage control is not one that causes single-, double-, or triple-spacing (blank, 0, or +), the line is not suppressed. Consequently, the first line at the top of the page is never suppressed.
 - Up to three `SUPPRESS SPACE` statements are allowed for each report.
 - The `SUPPRESS SPACE` action occurs after arithmetic and `DECIMALEEDIT` operations so that line suppression can be coded to appear as the result of a calculation or on lines that have a zero value in a field.

SPACEBEFORE POSnnn \bar{x}
SPACEAFTER POSnnn \bar{x}

Inserts a blank line before and/or after a specified line (nnn).

nnn -- Any 3-digit number between 001 and 132 that indicates the position where the x character is or is not found.

x -- Character (any printing or nonprinting character) that invokes the operation, if:

= x character is found.

* x character is not found.

These keywords override the NOSPACELINES and SUPPRESS SPACE keywords.

- If you try to insert two blank lines by following SPACEAFTER with SPACEBEFORE, only one will result.
- The keyword is effective only on single-spaced reports because it changes carriage control characters instead of creating actual blank lines.
- Up to three occurrences of each keyword are allowed for each report.

LINE SIZE=(nnn,nnn)
[,MOVEPAGENO]

The LINE SIZE=(nnn,nnn) part of this keyword allows you to reduce the size of a page from the size specified in the first nnn to that of the second nnn, where nnn is any position from 001–132. Fields beyond the truncated page size will not appear on the report.

The MOVEPAGENO part of this keyword is optional. When it is used, the page number (valid only for upper-right page positions) appears in the same relative position on the narrowed page.

Note: The report title is not automatically recentered by VISION:Builder when the LINE SIZE parameters are specified. To recenter the title on the narrowed page, the TITLE command operands should contain trailing blanks at the end of the text.

Diagnostic Messages

The following table lists the dynamic report line modification diagnostic messages:

Message	Explanation
>3 SEPCHAR	XREP allows a maximum of three separator characters between integers. This has been exceeded.
>13%/RAT.	A maximum of 13 PERCENT and/or RATIO keywords are allowed in one report. This limit has been exceeded.
=/1MISS.	The equal or not equal operator is invalid or missing for the keyword used.
A-DEC.>4	The number of places after the decimal point in the operand A picture of the PERCENT/RATIO keyword must not exceed four.
A-END>PGW	The last position of the operand A of the PERCENT/RATIO keyword is specified outside the print line width.
A-INTEG=0	No digits were found before the decimal point in the operand A picture of the PERCENT/RATIO keyword or the ROUND keyword.
A-INTEG>11	The number of digits before the decimal point in the operand A picture of the PERCENT/RATIO keyword must not exceed 11.
A-OP MISS.	The operand A of the PERCENT/RATIO keyword is missing.
A-PICT.END	The operand A picture of the PERCENT/RATIO keyword is not terminated by a right parenthesis. This message may also occur when the last character of the picture is the decimal point.
AADR>PGW	The positional indicator of the operand A of the PERCENT/RATIO keyword is outside the print line width for the keyword used.
AADR WRONG	The first positional indicator is not a 3-digit number for the keyword used. Leading zeroes must be supplied.
ADDR>PGW	The first positional indicator is outside the print line width.
ADDR WRONG	The first positional indicator is not a 3-digit number for the keyword used. Leading zeroes must be supplied.
ADR1>PGW	The first positional indicator of the SUPPRESS SPACE keyword is outside the print line width.

Message	Explanation
ADR1 WRONG	The first positional indicator of the SUPPRESS SPACE keyword is not a 3-digit number. Leading zeroes must be supplied.
ADR1>ADR2	The first positional indicator is greater than the second in the SUPPRESS SPACE keyword.
ADR2>PGW	The second positional indicator of the SUPPRESS SPACE keyword is outside the print line width.
ADR2 WRONG	The second positional indicator of the SUPPRESS SPACE keyword is not a 3-digit number. Leading zeroes must be supplied.
B-DEC>4	The number of places after the decimal point in the operand B picture of the PERCENT/RATIO keyword must not exceed four.
B-END>PGW	The last position of the operand B of the PERCENT/RATIO keyword is specified outside the print line width.
B-INTEG=0	No digits were found before the decimal point in the operand B of the PERCENT/RATIO keyword.
B-INTEG>11	The number of digits before the decimal point in the operand B picture of the PERCENT/RATIO keyword must not exceed 11.
B-OP MISS.	The operand B of the PERCENT/RATIO keyword is missing.
B-PICT.END	The operand B picture of the PERCENT/RATIO keyword is not terminated by a right parenthesis. This message may also occur when the last character of the picture is the decimal point.
BAD ASACHAR	The ASA character is not between 2 and 9 in the SKIP keyword.
BADR>PGW	The positional indicator of the operand B of the PERCENT/RATIO keyword is outside the print line width.
BADR WRONG	The positional indicator of the operand B of the PERCENT/RATIO keyword is not a 3-digit number. Leading zeroes must be supplied.
C-DEC MISS.	No decimal places were found after the decimal point in the operand C picture of the PERCENT/RATIO keyword.

Message	Explanation
C-DEC>4	The number of places after the decimal point in the operand C picture of the PERCENT/RATIO keyword must not exceed four.
C-END>PGW	The last position of the operand C of the PERCENT/RATIO keyword is specified outside the print line width.
C-INTEG=0	No digits were found before the decimal point in the operand C of the PERCENT/RATIO keyword.
C-INTEG>9	The number of digits before the decimal point in the operand C picture of the RATIO keyword must not exceed nine.
C-INTEG>11	The number of digits before the decimal point in the operand C picture of the PERCENT keyword must not exceed 11.
C-OP MISS.	The operand C of the PERCENT/RATIO keyword is missing.
C-PICT.END	This message may occur for any of the following conditions: <ul style="list-style-type: none"> ■ The operand C picture of the PERCENT/ RATIO keyword is not terminated by a right parenthesis. ■ The last character of the picture is the decimal point. ■ The last character of the operand C picture of the PERCENT keyword is not a percent sign (%). ■ Separating characters were found in the operand C picture of the PERCENT/RATIO keyword.
CADR>PGW	The positional indicator of the operand C of the PERCENT/RATIO keyword is outside the print line width.
CADR WRONG	The positional indicator of the operand C of the PERCENT/RATIO keyword is not a 3-digit number. Leading zeroes must be supplied.
CDEC UNNUM	Nonnumeric characters were found in the operand C picture of the PERCENT keyword.
MORE THAN 3	Only three keywords of this type are allowed per report.
NO.NOT NUM	The value in the operand of the TITLE LINENO keyword is not one or two digits.

Message	Explanation
NOMORE CORE	Too many PERCENT, RATIO, or ROUND calculations have been specified. Core is exceeded.
OP.2 MISS.	The second positional indicator is missing for the keyword used.
OP.3 WRONG	SIZE in the LINE SIZE keyword is not followed by a delimiter or MOVEPAGENO.
PARAMREPEAT	Only one parameter of this type is allowed per report.
SHORT LINE	The reduced line size specified in the LINE SIZE keyword is not long enough to contain the page number.
SIZE L1<L2	The modified page width must be less than the input page width.
SZ.1>PGW	The input page width is greater than the print line width.
SZ.1 WRONG	The input page width operand is not a 3-digit number in the LINE SIZE keyword. Leading zeroes must be supplied.
SZ.2>PGW	The modified page width is greater than the print line width.
SZ.2 WRONG	The modified page width operand is not a 3-digit number in the LINE SIZE keyword. Leading zeroes must be supplied.

Example

```
REPORT VENDOR DUEDATE AMOUNT
  TITLE 'ACCOUNTS RECEIVABLE'
  TITLE 'AGING REPORT'
  ORDER BY VENDOR DUEDATE
  GROUP BY VENDOR
  XREP 'NOSPACELINES'
END REPORT
```

The XREP command in this example suppresses the output of all blank lines in this report.

Subfile Output Command Group

Subfile output is specified using the EXTRACT command. There are four variations of the EXTRACT command as follows:

Command	Function
EXTRACT FILE	Extract to a file identified by a FILE SUBFn command
EXTRACT DDNAME	Extract to a sequential or VSAM file identified by DD Name
EXTRACT DBDNAME	Extract to an IMS database identified by DBD Name
EXTRACT TABLE	Extract to a DB2 table identified by Table Name

The EXTRACT command may be embedded inside a procedure block, and a procedure block can contain many EXTRACT commands. When embedded inside a procedure, an extract will be invoked based upon the logic flow of the procedure. When present outside of a procedure, an extract will be invoked based upon the TYPE keyword operand.

The EXTRACT FILE command requires that a corresponding FILE SUBFn command appear in the run control section of the application. Subfiles created in this way are referred to as traditional subfiles. The EXTRACT DDNAME, EXTRACT DBDNAME, and EXTRACT TABLE commands do not require a corresponding FILE SUBFn command and are referred to as extended subfiles.

EXTRACT FILE Command

```
[extract-name:] EXTRACT FILE subfile-name,

    {[COLUMNS] field-name ... | ENTIRE qual-char},
    [KEYS key-field ...],
    [VARCHARMAX NOPREFIX varchar-field1 ...],
    [VARCHARMAX WITHPREFIX varchar-field2 ...],
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [ABBREVIATED],
    [RECFM {FIXED | VARIABLE | UNDEFINED | KEYVSAM |
           ENTRYVSAM | DLI | HDAM | DB2 | PACKED}],
    [BLKSIZE block-size],
    [AUTODEF],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ |
          TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
          EOF | EOFPLUS}],
    [INFO 'text']
```

The EXTRACT FILE command is used to specify subfile output to a file declared by a FILE SUBF_n command in the Run Control section. The EXTRACT command may be included inside a procedure block or may appear outside of any procedure block. When included inside of a procedure block, its invocation will be controlled by the procedure. When outside of a procedure block, its invocation will be controlled by the use of the TYPE keyword.

<i>extract-name</i> :	Specifies the name of the extract. Naming an extract is optional and is only required when the EXTRACT command is not inside a procedure block and TYPE is specified as SUBROUTINE.
FILE <i>subfile-name</i>	Specifies the name that matches the subfile name on a FILE SUBF _n to which this subfile will be output. More than one EXTRACT command may reference the same subfile name.
COLUMNS <i>field-name</i> ... COLUMN	Specifies a list of one or more qualified field names that make up the subfile record in the order that they are listed. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the ENTIRE keyword.
ENTIRE <i>qual-char</i>	Specifies the qualifier for the file whose entire record is to become the subfile record. This keyword is mutually exclusive with the ITEMS keyword.

KEYS *key-field* ...
KEY

Specifies the qualified names of one or more fields that have been included with the ITEMS keyword operand list that are to be identified as key fields in the created file definition when the AUTODEF keyword is specified. The field may be a partial field using the PF function notation and must match the partial field specifications for the field in the ITEMS keyword operand list. The first field in the list becomes key field 1, the second key field 2, and so on. This keyword is only meaningful when the ITEMS keyword is also specified.

VARCHARMAX NOPREFIX *varchar-field1* ...
VMAX NOP

Specifies the qualified names of one or more varchar fields included in the ITEMS keyword operand list whose output size will be the maximum length of the varchar field. The field may be a partial field using the PF function notation. The field will be filled with trailing blanks up to the output size. No prefix indicating the number of significant characters in the varchar field will be output.

VARCHARMAX WITHPREFIX *varchar-field2* ...
VMAX WITHP

Specifies the qualified names of one or more varchar fields included in the ITEMS keyword list whose output size will be the maximum length of the varchar field. The field may be a partial field using the PF function notation. The field will be filled with trailing blanks up to the output size. A 2-byte prefix will precede the field and will contain the number of significant characters in the field.

SELECTIONCONTROL {YES | NO}
SELCNTL

Specifies whether or not all occurrences of repeated segments in a hierarchical structure are to be examined.

- YES -- Specifies that only the first occurrence of a repeated segment within each record that meets the selection qualification will be selected.
- NO -- Specifies that all occurrences of repeated segments within each record that meet the selection qualification will be selected. This is the default if the SELECTIONCONTROL keyword is omitted.

MAXITEMS *number*

Specifies the maximum number of records that will be selected for this subfile. Specify a number from 1 to 9999 to limit the number of records that will be selected for this subfile. If this keyword is omitted, the number of records that will be selected is unlimited.

ABBREVIATED
ABBREV

Specifies whether repeated selections of the same data will be output only once. This use of this keyword is only meaningful when one or more of the input files contain data in a hierarchical structure wherein the segments have multiple branches or when more than one standard coordinated file (see the FILE CORDn command) is used. These coordinated files may be viewed as a hierarchical structure with multiple branches.

Note: In earlier versions of VISION:Builder, this functionality was achieved through a specification called Parallel Looping. The use of ABBREVIATED here limits the action of Parallel Looping to output operations only.

RECFM *record-format*

Specifies the access method and record format for the subfile. If this keyword is omitted, the variable length format sequential file will be used. Valid specifications for this keyword are:

- **FIXED** -- Specifies that the subfile will be a fixed length record format sequential file.
- **VARIABLE** -- Specifies that the subfile will be a variable length record format sequential file.
- **UNDEFINED** -- Specifies that the subfile will be an undefined length record format sequential file.
- **KEYVSAM** -- Specifies that the subfile will be a Key Sequenced VSAM file.
- **ENTRYVSAM** -- Specifies that the subfile will be an Entry Sequenced VSAM file.
- **DLI** -- Specifies that the subfile will be an IMS database using HISAM, HIDAM, or HSAM. When this keyword is specified, the ENTIRE keyword must also be specified.
- **HDAM** -- Specifies that the subfile will be an IMS database using HDAM. When this keyword is specified, the ENTIRE keyword must also be specified.
- **DB2** -- Specifies that the subfile will be a DB2 table. When this keyword is specified, the ITEMS keyword must also be specified.
- **PACKED** -- Specifies that the subfile will be a variable length record file sequential file with record compression.

BLKSIZE <i>block-size</i>	Specifies the maximum size of a data block for the VARIABLE, UNDEFINED, and PACKED record formats. Specifies the blocking factor (records per block) for the FIXED record format. This keyword should not be specified for the KEYVSAM, ENTRYVSAM, DLI, HDAM, and DB2 record formats. If this keyword is omitted for the sequential file record formats, operating system defaults will be used. It is recommended that this keyword only be used in special circumstances.
AUTODEF	Specifies that file definition statements will be generated for the fields contained in this subfile. The file definition name will be the same as the subfile-name. A FILE SUBFn command with the AUTODEF keyword must be present in the Run Control section of the application when this keyword is specified.
TYPE <i>invocation-event</i>	<p>Specifies the event within the application processing cycle at which the extract output will be invoked. If the EXTRACT command is inside of a procedure block, this keyword should be omitted because the PROC statement will control the invocation event. If this keyword is omitted and the command is not inside of a procedure block, the extract output will be invoked during each iteration of the master file read and standard coordination cycles. Valid invocation event specifications are as follows:</p> <ul style="list-style-type: none">■ NORMAL -- Specifies that the subfile output will be invoked during each iteration of master file read and standard coordination cycles.■ SUBROUTINE or SUB -- Specifies that the subfile output will only be invoked when explicitly called by a procedural CALL statement.■ INIT -- Specifies that the report subfile will be invoked only one time after the master and coordinated files have been opened but before any records from these files have been processed.

- **PRE_MASTER_READ** -- Specifies that the subfile output will be invoked just prior to reading the next master file record.
- **TYPE1** -- Specifies that the subfile output will be invoked after each transaction record has been read.
- **TYPE2** -- Specifies that the subfile output will be invoked after the transaction file and the master files have been aligned.
- **TYPEM** -- Specifies that the subfile output will be invoked after the master file record has been updated but before first round coordination.
- **TYPE3** -- Specifies that the subfile output will be invoked after the master file record has been updated but after first round coordination.
- **TYPE4** -- Specifies that the subfile output will be invoked whenever a transaction record is rejected, either explicitly by procedural code or implicitly by transaction validation controls.
- **EOF** -- Specifies that the subfile output will be invoked only once after the transaction file, master file, and all coordinated files have reached end of file.
- **EOFPLUS** -- Specifies that the subfile output will be invoked as in **NORMAL** and additionally as in **EOF** above.
- Specifies descriptive text that may be used to annotate the subfile. The text cannot be longer than 28 characters.

INFO '*text*'

Examples

```
EXTRACT FILE SUBOUT1,  
  COLUMNS EMPNO EMPNAME PHONENO
```

The above example shows how subfile output can be coded.

```
PROC TYPE TRAN3  
  CALL SUBFILE EXTR1  
END PROC  
;  
EXTR1: EXTRACT FILE SUBOUT2, ENTIRE X, TYPE SUBROUTINE
```

The above example shows how subfile output can be coded as a subroutine and called from a procedure.

```
PROC  
  IF HIREDATE LT 01011990  
    EXTRACT FILE SUBOUT3, RECFM KEYVSAM, BLKSIZE 2300,  
      COLUMNS WORKDEPT, 1.DEPTNAME, EMPNAME,  
      AUTODEF  
  END IF  
END PROC
```

The above example shows how subfile output may be embedded inside of a procedure block.

EXTRACT DDNAME Command

```
[extract-name:] EXTRACT DDNAME ddname,

    {[COLUMNS] field-name ... | ENTIRE qual-char},
    [KEYS key-field ...],
    [VARIABLE var-field ...],
    [VARIABLE WITHPREFIX var-field ...],
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [ABBREVIATED],
    [RECFM {FIXED | VARIABLE | UNDEFINED |
           KEYVSAM | ENTRYVSAM | PACKED}],
    [BLKSIZE block-size],
    [AUTODEF],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ
          | TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
          EOF | EOFPLUS}],
    [INFO 'text']
```

The EXTRACT DDNAME command is used to specify subfile output to the file identified with the DDNAME keyword. The EXTRACT command may be included inside a procedure block or may appear outside of any procedure block. When included inside of a procedure block, its invocation will be controlled by the procedure. When outside of a procedure block, its invocation will be controlled by the use of the TYPE keyword.

<i>extract-name</i> :	Specifies the name of the extract. Naming an extract is optional and is only required when the EXTRACT command is not inside a procedure block and TYPE is specified as SUBROUTINE.
DDNAME <i>ddname</i>	Specifies the DD name corresponding to the file to which the subfile will be output. Only one EXTRACT command should reference this DD name.
COLUMNS <i>field-name</i> ... COLUMN	Specifies the qualified names of one or more fields that make up the subfile record in the order that they are listed. The field may be a partial field using the PF function notation. This keyword is mutually exclusive with the ENTIRE keyword.
ENTIRE <i>qual-char</i>	Specifies the qualifier for the file whose entire record is to become the subfile record. This keyword is mutually exclusive with the ITEMS keyword.

KEYS *key-field* ...
KEY

Specifies the qualified names of one or more fields that have been included with the ITEMS keyword operand list that are to be identified as key fields in the created file definition when the AUTODEF keyword is specified. The field may be a partial field using the PF function notation. The first field in the list becomes key field 1, the second key field 2, and so on. This keyword is only meaningful when the ITEMS keyword is also specified.

VARCHARMAX NOPREFIX *varchar-field1* ...
VMAX NOP

Specifies the names of one or more varchar fields included in the ITEMS keyword list whose output size will be the maximum length of the varchar field. The field will be filled with trailing blanks up to the output size. No prefix indicating the number of significant characters in the varchar field will be output.

VARCHARMAX WITHPREFIX *varchar-field2* ...
VMAX WITHP

Specifies the names of one or more varchar fields included in the ITEMS keyword list whose output size will be the maximum length of the varchar field. The field will be filled with trailing blanks up to the output size. A 2-byte prefix will precede the field and will contain the number of significant characters in the field.

SELECTIONCONTROL {YES | NO}
SELCNTL

Specifies whether or not all occurrences of repeated segments in a hierarchical structure are to be examined.

- YES -- Specifies that only the first occurrence of a repeated segment within each record that meets the selection qualification will be selected.
- NO -- Specifies that all occurrences of repeated segments within each record that meet the selection qualification will be selected. This is the default if the SELECTIONCONTROL keyword is omitted.

MAXITEMS <i>number</i>	Specifies the maximum number of records that will be selected for this subfile. Specify a number from 1 to 9999 to limit the number of records that will be selected for this subfile. If this keyword is omitted, the number of records that will be selected is unlimited.
ABBREVIATED ABBREV	<p>Specifies whether repeated selections of the same data will be output only once. This use of this keyword is only meaningful when one or more of the input files contain data in a hierarchical structure wherein the segments have multiple branches or when more than one standard coordinated file (see the FILE CORDn command) is used. These coordinated files may be viewed as a hierarchical structure with multiple branches.</p> <p>Note: In earlier versions of VISION:Builder, this functionality was achieved through a specification called Parallel Looping. The use of ABBREVIATED here limits the action of Parallel Looping to output operations only.</p>
RECFM <i>record-format</i>	<p>Specifies the access method and record format for the subfile. If this keyword is omitted, the variable length format sequential file will be used. Valid specifications for this keyword are:</p> <p>FIXED -- Specifies that the subfile will be a fixed length record format sequential file.</p> <p>VARIABLE -- Specifies that the subfile will be a variable length record format sequential file.</p> <p>UNDEFINED -- Specifies that the subfile will be an undefined length record format sequential file.</p> <p>KEYVSAM -- Specifies that the subfile will be a Key Sequenced VSAM file.</p> <p>ENTRYVSAM -- Specifies that the subfile will be an Entry Sequenced VSAM file.</p> <p>PACKED -- Specifies that the subfile will be a variable length record sequential file with record compression.</p>

BLKSIZE <i>block-size</i>	Specifies the maximum size of a data block for the VARIABLE, UNDEFINED, and PACKED record formats. Specifies the blocking factor (records per block) for the FIXED record format. This keyword should not be specified for the KEYVSAM and ENTRYVSAM record formats. If this keyword is omitted for the non-VSAM record formats, operating system defaults will be used. It is recommended that this keyword only be used for special circumstances.
AUTODEF	Specifies that file definition statements will be generated for the fields contained in this subfile. The file definition name will be the same as the ddname. A FILE SUBF <i>n</i> command with the AUTODEF keyword must be present in the Run Control section of the application when this keyword is specified.
TYPE <i>invocation-event</i>	<p>Specifies the event within the application processing cycle at which the extract output will be invoked. If the EXTRACT command is inside of a procedure block, this keyword should be omitted because the PROC statement will control the invocation event. If this keyword is omitted and the command is not inside of a procedure block, the extract output will be invoked during each iteration of the master file read and standard coordination cycles. Valid invocation event specifications are as follows:</p> <p>NORMAL -- Specifies that the subfile output will be invoked during each iteration of master file read and standard coordination cycles.</p> <p>SUBROUTINE or SUB -- Specifies that the subfile output will only be invoked when explicitly called by a procedural CALL statement.</p> <p>INIT -- Specifies that the report subfile will be invoked only one time after the master and coordinated files have been opened but before any records from these files have been processed.</p> <p>PRE_MASTER_READ -- Specifies that the subfile output will be invoked just prior to reading the next master file record.</p>

TYPE1 -- Specifies that the subfile output will be invoked after each transaction record has been read.

TYPE2 -- Specifies that the subfile output will be invoked after the transaction file and the master files have been aligned.

TYPEM -- Specifies that the subfile output will be invoked after the master file record has been updated but before first round coordination.

TYPE3 -- Specifies that the subfile output will be invoked after the master file record has been updated but after first round coordination.

TYPE4 -- Specifies that the subfile output will be invoked whenever a transaction record is rejected, either explicitly by procedural code or implicitly by transaction validation controls.

EOF -- Specifies that the subfile output will be invoked only once after the transaction file, master file, and all coordinated files have reached end of file.

EOFPLUS -- Specifies that the subfile output will be invoked as in NORMAL and additionally as in EOF above.

INFO *'text'*

Specifies descriptive text that may be used to annotate the subfile. The text cannot be longer than 28 characters.

Examples

```
EXTRACT DDNAME FILE1, COLUMNS CUSTNO CUSTNAME BALANCE
```

```
EXTRACT DDNAME PARTLIST, COLUMNS PARTNO DESCRIPT PRICE,  
VARCHARMAX NOPREFIX DESCRIPT, AUTODEF
```

The above examples show extended subfile output to a specific DD name.

EXTRACT DBDNAME Command

```
[extract-name:] EXTRACT DBDNAME dbdname,  
  
    ENTIRE qual-char,  
    [{DLI | HDAM}],  
    [SELECTIONCONTROL {YES | NO}],  
    [MAXITEMS number],  
    [AUTODEF],  
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE_MASTER_READ |  
          TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |  
          EOF | EOFPLUS}],  
    [INFO 'text']
```

The EXTRACT DBDNAME command is used to specify subfile output to an IMS database identified with the DBDNAME keyword. The EXTRACT command may be included inside a procedure block or may appear outside of any procedure block. When included inside of a procedure block, its invocation will be controlled by the procedure. When outside of a procedure block, its invocation will be controlled by the use of the TYPE keyword.

<i>extract-name</i> :	Specifies the name of the extract. Naming an extract is optional and is only required when the EXTRACT command is not inside a procedure block and TYPE is specified as SUBROUTINE.
DBDNAME <i>dbdname</i>	Specifies the name of the IMS database definition (DBD) that identifies the database into which the selected records are to be inserted.
ENTIRE <i>qual-char</i>	Specifies the qualifier of the file whose entire records are to be inserted into the IMS database. Only entire records may be output into an IMS subfile.
DLI	Specifies that the IMS database is a HISAM, HIDAM, or HSAM database. This keyword is mutually exclusive with the HDAM keyword and is the default if neither the DLI nor HDAM keyword is specified.
HDAM	Specifies that the IMS database is a HDAM database. This keyword is mutually exclusive with the DLI keyword.

SELECTIONCONTROL {YES | NO}

SELCNTL

Specifies whether or not all occurrences of repeated segments in a hierarchical structure are to be examined.

YES -- Specifies that only the first occurrence of a repeated segment within each record that meets the selection qualification will be selected.

NO -- Specifies that all occurrences of repeated segments within each record that meet the selection qualification will be selected. This is the default if the SELECTIONCONTROL keyword is omitted.

MAXITEMS *number*

Specifies the maximum number of records that will be selected for this subfile. Specify a number from 1 to 9999 to limit the number of records that will be selected for this subfile. If this keyword is omitted, the number of records that will be selected is unlimited.

AUTODEF

Specifies that file definition statements will be generated for the fields contained in this subfile. The file definition name will be the same as the dbdname. A FILE SUBFn command with the AUTODEF keyword must be present in the Run Control section of the application when this keyword is specified.

TYPE *invocation-event*

Specifies the event within the application processing cycle at which the extract output will be invoked. If the EXTRACT command is inside of a procedure block, this keyword should be omitted because the PROC statement will control the invocation event. If this keyword is omitted and the command is not inside of a procedure block, the extract output will be invoked during each iteration of the master file read and standard coordination cycles. Valid invocation-event specifications are as follows:

NORMAL -- Specifies that the subfile output will be invoked during each iteration of master file read and standard coordination cycles.

SUBROUTINE or SUB -- Specifies that the subfile output will only be invoked when explicitly called by a procedural CALL statement.

INIT -- Specifies that the report subfile will be invoked only one time after the master and coordinated files have been opened but before any records from these files have been processed.

PRE_MASTER_READ -- Specifies that the subfile output will be invoked just prior to reading the next master file record.

TYPE1 -- Specifies that the subfile output will be invoked after each transaction record has been read.

TYPE2 -- Specifies that the subfile output will be invoked after the transaction file and the master files have been aligned.

TYPEM -- Specifies that the subfile output will be invoked after the master file record has been updated but before first round coordination.

TYPE3 -- Specifies that the subfile output will be invoked after the master file record has been updated but after first round coordination.

TYPE4 -- Specifies that the subfile output will be invoked whenever a transaction record is rejected, either explicitly by procedural code or implicitly by transaction validation controls.

EOF -- Specifies that the subfile output will be invoked only once after the transaction file, master file, and all coordinated files have reached end of file.

EOFPLUS -- Specifies that the subfile output will be invoked as in NORMAL and additionally as in EOF above.

INFO *'text'*

Specifies descriptive text that may be used to annotate the subfile. The text cannot be longer than 28 characters.

Examples

```
EXTRACT DBDNAME NEWDBD, ENTIRE O, HDAM
```

The above example shows extended subfile output to an IMS database.

EXTRACT TABLE Command

```
[extract-name:] EXTRACT TABLE "authid.tablename",
    {CREATE | DELETE | INSERT | DROP},
    [TABLESPACE tablespace-name],
    [DATABASE database-name],
    COLUMNS {field-name | PF(field-name start length) ...},
    [KEYS {key-field | PF(key-field start length) ...},
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [ABBREVIATED],
    [AUTODEF],
    [DEFNAME definition-name],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ |
          TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
          EOF | EOFPLUS}],
    [INFO 'text']
```

The EXTRACT TABLE command is used to specify subfile output to a DB2 table identified with the TABLE keyword. The EXTRACT command may be included inside a procedure block or may appear outside of any procedure block. When included inside of a procedure block, its invocation will be controlled by the procedure. When outside of a procedure block, its invocation will be controlled by the use of the TYPE keyword.

<i>extract-name</i> :	Specifies the name of the extract. Naming an extract is optional and is only required when the EXTRACT command is not inside a procedure block and TYPE is specified as SUBROUTINE.
TABLE " <i>authid.tablename</i> "	Specifies the name of the DB2 table and optionally, the authorization id to be assigned to the table. If authorization id is specified, it must be followed with a period (.) and the entire string of authorization id and table name enclosed in quotation marks ("). If authorization id is omitted, the default authorization id determined by DB2 conventions will be used.
CREATE	Specifies that the DB2 table is to be created before any rows are inserted. The identifiers specified with the TABLESPACE and DATABASE keywords will be used in the table creation process.

DELETE	Specifies that any existing rows in the DB2 table will be deleted prior to any new rows being inserted.
INSERT	Specifies that new rows are to be inserted into the existing table. Note that the table may contain duplicate data after processing is complete when this keyword is specified. Duplicate data may be avoided by creating an index with the unique keys attribute for the table. If this is the case and processing should continue after a duplicate row is encountered, the CONTROL command should specify the CONTINUE keyword.
DROP	Specifies that the existing DB2 table is to be dropped and a new one created before any rows are inserted. The identifiers specified with the TABLESPACE and DATABASE keywords will be used in the table creation process.
TABLESPACE <i>tablespace-name</i>	Specifies the name of the DB2 table space in which the table is to be created. This keyword is only meaningful when CREATE or DROP is specified. If this keyword is omitted when CREATE or DROP is specified, the DB2 defaults will be used.
DATABASE <i>database-name</i>	Specifies the name of the DB2 database in which the table is to be created. This keyword is only meaningful when CREATE or DROP is specified. If this keyword is omitted when CREATE or DROP is specified, the DB2 defaults will be used.
COLUMNS <i>field-name</i> ... COLUMN	Specifies a list of one or more qualified names of fields that will become the columns in the DB2 table in the order that they are listed. The field may be a partial field using the PF function notation.
KEYS <i>key-field</i> ... KEY	Specifies the names of one or more field names that have been included with the ITEMS keyword list, and which are to be identified as key fields in the created file definition when the AUTODEF keyword is specified. The first field in the list becomes key field 1, the second key field 2, and so on.

SELECTIONCONTROL {YES | NO}
SELCNTL

Specifies whether or not all occurrences of repeated segments in a hierarchical structure are to be examined.

YES -- Specifies that only the first occurrence of a repeated segment within each record that meets the selection qualification will be selected.

NO -- Specifies that all occurrences of repeated segments within each record that meet the selection qualification will be selected. This is the default if the SELECTIONCONTROL keyword is omitted.

MAXITEMS *number*

Specifies the maximum number of records that will be selected for this subfile. Specify a number from 1 to 9999 to limit the number of records that will be selected for this subfile. If this keyword is omitted, the number of records that will be selected is unlimited.

ABBREVIATED
ABBREV

Specifies whether repeated selections of the same data will be output only once. The use of this keyword is only meaningful when one or more of the input files contain data in a hierarchical structure wherein the segments have multiple branches or when more than one standard coordinated file (see the FILE CORDn command) is used. These coordinated files may be viewed as a hierarchical structure with multiple branches.

If this keyword is specified, an output record will only be produced the first time that each unique combination of different repeating segments is examined and the qualifying criteria are met. If this keyword is omitted, an output record will be produced each time any combination of different repeating segments is examined and the qualifying criteria are met.

Note: In earlier versions of VISION:Builder, this functionality was achieved through a specification called Parallel Looping. The use of ABBREVIATED here limits the action of Parallel Looping to output operations only.

AUTODEF	Specifies that file definition statements will be generated for the fields contained in this subfile. A FILE SUBFn command with the AUTODEF keyword must be present in the Run Control section of the application when this keyword is specified.
DEFNAME <i>definition-name</i>	Specifies the file definition name to be used for the file definition created by specifying the AUTODEF command. If this keyword is omitted and the AUTODEF keyword is specified, a default definition name of X\$SUBnnn will be used where nnn is a uniquely assigned number.

TYPE *invocation-event*

Specifies the event within the application processing cycle at which the extract output will be invoked. If the EXTRACT command is inside of a procedure block, this keyword should be omitted because the PROC statement will control the invocation event. If this keyword is omitted and the command is not inside of a procedure block, the extract output will be invoked during each iteration of the master file read and standard coordination cycles. Valid invocation-event specifications are as follows:

NORMAL -- Specifies that the subfile output will be invoked during each iteration of master file read and standard coordination cycles.

SUBROUTINE or SUB -- Specifies that the subfile output will only be invoked when explicitly called by a procedural CALL statement.

INIT -- Specifies that the report subfile will be invoked only one time after the master and coordinated files have been opened but before any records from these files have been processed.

PRE_MASTER_READ -- Specifies that the subfile output will be invoked just prior to reading the next master file record.

TYPE1 -- Specifies that the subfile output will be invoked after each transaction record has been read.

TYPE2 -- Specifies that the subfile output will be invoked after the transaction file and the master files have been aligned.

TYPEM -- Specifies that the subfile output will be invoked after the master file record has been updated but before first round coordination.

TYPE3 -- Specifies that the subfile output will be invoked after the master file record has been updated but after first round coordination.

TYPE4 -- Specifies that the subfile output will be invoked whenever a transaction record is rejected, either explicitly by procedural code or implicitly by transaction validation controls.

EOF -- Specifies that the subfile output will be invoked only once after the transaction file, master file, and all coordinated files have reached end of file.

EOFPLUS -- Specifies that the subfile output will be invoked as in NORMAL and additionally as in EOF above.

INFO *'text'*

Specifies descriptive text that may be used to annotate the subfile. The text cannot be longer than 28 characters.

Examples

```
EXTRACT TABLE "PAYROLL.NEWTABLE", CREATE,  
TABLESPACE TEMPSpace, DATABASE PAYDB,  
COLUMNS HIREDATE BEGIN_SALARY CURRENT_SALARY,  
KEYS HIREDATE,  
AUTODEF, DEFNAME INCRATE
```

The above example shows extended subfile output to a DB2 table.

This chapter contains examples of ASL that show how the language is used in the context of an entire procedure or application. The purpose of these examples is to illustrate how the language is used to perform a variety of operations that use the various features of the language.

Example 1

This example converts feet to centimeters. The program converts the value in field DISTANCE to centimeters and stores the result back into the field DISTANCE in the logical record. There is no change to the physical data.

```
; Convert distance in units of feet to distance in unit of
; centimeters.
;
LET DISTANCE = DISTANCE * 30.480 WITH ROUNDING
```

The WITH ROUNDING clause rounds the result to the nearest centimeter.

Because of the set operation of ASL, this one line procedure transforms all instances of the field DISTANCE in the database.

Example 2

This example reports only employee records for employees whose salaries are less than \$50,000.

```
; Bypass all records for salaries of $50,000 and greater.
;
IF SALARY >= 50000
  TRANSFER TO NEXT_MASTER
END
```

All records for employees whose salaries are \$50,000 or greater are ignored.

Example 3

This example reads a parameter record and places entries from the parameter record into working storage fields. If you do not provide a parameter record, the example writes the message text "NO PARAMETER RECORD" on the report ERRLIST and terminates the run by forcing end of file on the master file.

```
; Read parameter record into working storage.
;
IF FIND(9.PARMDATA)
```

```

      LET W.PARM1 = PF(9.PARMS 1 8)           ;First parameter field
      LET W.PARM2 = PF(9.PARMS 10 4)        ;Second parameter field
ELSE
  LET T.ERRMSG = 'NO PARAMETER RECORD'
  CALL REPORT ERRLIST                       ;Generate error report
  LET PF(F.EOF 1 1) = 'E'                 ;Force End of File
END

```

The example:

- Reads the parameter record from file 9 using the FIND function.
- The FIND function returns a FALSE command if no record is available (that is, the file is empty or at end of file).
- Uses partial fielding of the field PARMS (in file 9) to isolate the parameters.
- Terminates if no parameter record is found by placing the code E into the first position of the EOF flag field. The first position controls the master file. When the master file reaches the end of file, there are no secondary file coordinations and the program terminates.

Example 4

This example illustrates the use of qualifiers to reference data from multiple files.

```

; Compute total by adding amounts from the file for each quarter.
;
LET T.TOTAL = 1.QTRTOT + 2.QTRTOT + 3.QTRTOT + 4.QTRTOT

```

Note that this example uses the qualifiers to indicate the file from which each quarter's totals are obtained.

The field names within each of the coordinated files are QTRTOT. Each of the files can either use the same definition or different definitions. The qualifier identifies from which of the coordinated files the data is obtained. The temporary field TOTAL uses the qualifier T to identify that it is a computed field. The field is available to all other procedures.

Example 5

This example illustrates the use of the FIND function for performing a file browse.

```

; ***** File Browse Example *****
;
; Report all customers whose customer name begins with 'B'.
; This assumes that the customer name is the primary key of the
; file.
;
IF FIND(1.CUSTOMER WHERE 1.CUSTNAME GE 'B')
  DO UNTIL NOT FIND(1.CUSTOMER)
    IF PF(CUSTNAME 1 1) EQ 'B'
      CALL REPORT LISTCUST
    ELSE
      LEAVE
  END
END
END

```

In this example, the first statement (IF) reads (FIND) the first record of the file whose key (CUSTNAME) begins with B or greater.

- If no record exists for this condition, the program does not produce report output and terminates the procedures.
- If the first FIND statement reads a record, the customer name (CUSTNAME) is examined to see if it begins with a B.
- If not, the LEAVE statement exits the DO loop and the procedure terminates. Otherwise, the report output is produced and the loop continues by reading (FIND) the next sequential record from the file. The UNTIL NOT FIND clause fails and the loop terminates when the end of the file is reached or the customer name is greater than B.

Note that the program evaluates the UNTIL clause only after the loop executes at least one time.

Example 6

This example is of a GDBI mapping procedure for an ADABAS database.

```
; ***** Root-Level Mapping Procedure for ADABAS *****  
;  
IF F.COMMAND = 'GETFIRST'  
  CALL MODULE INNXXEP T.LOCATE STARTSEG T.RESCODE  
  IF T.RESCODE <> 0 OR F.CSTATUS <> ' '  
    LET F.MSTATUS = 'STOP10'  
    RETURN  
  END  
END  
CALL INNXXEP T.SEQREAD STARTSEG T.RESCODE  
IF F.CSTATUS = ' '  
  DO CASE  
    CASE WHERE T.RESCODE = 0  
      T.BCUSED = 0  
      RETURN  
    CASE WHERE T.RESCODE = -1  
      LET F.MSTATUS = 'NFOUND'  
      RETURN  
  END  
END  
LET F.MSTATUS = 'STOP20'
```

This procedure retrieves a root segment from the ADABAS database manager and places it into the logical record in field STARTSEG. Based on the flag field COMMAND, the procedure either retrieves the root segment starting from a particular key value or from the beginning of the database.

If the COMMAND flag field contains GETFIRST, the CALL statement interfaces to the INNXXEP module in ADABAS to set the position of the database to the root segment that matches the key value contained in T.LOCATE.

- If the CALL statement is not successful, a value of STOP10 is placed into the MSTATUS flag field and the procedure terminates.
- If the CALL statement is successful or if the COMMAND flag field does not contain GETFIRST, the procedure continues with the second CALL statement that interfaces to the INNXXEP module in ADABAS to actually retrieve the root

segment from the current position and return the data in the STARTSEG field. If the second CALL statement is successful, the DO CASE determines if ADABAS found a segment.

- The program sets the field T.BCUSED to zero if a segment is retrieved.
- The program sets the MSTATUS flag field to NFOUND if a segment is not retrieved. In either case, the RETURN statements terminate the procedure.
- If the second CALL statement is not successful, the program places a value of STOP20 into the MSTATUS flag field and the procedure terminates.

Example 7

This example isolates arguments from a text string.

```
;ISOLATE ARGUMENTS FROM A TEXT STRING
;
; INPUT IS T.TEXT, T.START, T.LENGTH
; OUTPUT IS T.ARGST (START OF ARGUMENT)
;           T.ARGL (LENGTH OF ARGUMENT)
;           T.START AND T.LENGTH ARE READY TO CONTINUE THE
;           REMAINDER OF THE SCAN.
; SETTINGS OF ZERO ARE ERROR CONDITIONS
;
LET F.LSTART = T.START           ;SET SCAN START
LET F.LNUMBER = T.LENGTH        ;SET SCAN LENGTH
;
IF SCAN(PF(T.TEXT LS LN) FOR P'z') ;SCAN FOR ALPHA
  LET T.ARGST = F.MSTART        ;SAVE ARG START
  LET T.LENGTH = F.RNUMBER+F.MNUMBER ;SET TO SCAN REMAINDER
  ;                               OF TEXT
ELSE
  LET T.ARGST = 0                ;NO ALPHA SO SET ARG
  ;                               PARAMETERS
  LET T.ARGL = 0                 ;TO ZERO
  RETURN
END
;
LET F.LSTART = T.ARGST          ;SET SCAN START
LET F.LNUMBER = T.LENGTH        ;SET SCAN LENGTH
IF SCAN(PF(T.TEXT LS LN) FOR P'B') ;SCAN FOR BLANK
  LET T.ARGL = F.LNUMBER        ;ARG LENGTH
  LET T.START = F.MSTART        ;SET NEW START
  LET T.LENGTH = F.RNUMBER + F.MNUMBER ;SET REMAINDER OF SCAN
ELSE
  LET T.ARGL = T.LENGTH          ;MUST BE END OF FIELD
  LET T.LENGTH = 0              ;SET FOR NO MORE TO SCAN
END
```

In this procedure, the SCAN function scans first for an alpha character, as indicated by the pattern P'z' in the first SCAN function. If the program finds an argument, it saves the starting position in the text line; otherwise, the procedure returns and indicates that no argument was found. After an argument is found, the second SCAN function looks for a blank by using the pattern P'B', which indicates the end of the argument.

This routine is used as a subroutine. The calling routine initializes the three temporary fields TEXT, START, and LENGTH. TEXT contains the text to be scanned. START contains the start of the scan. LENGTH contains the length to scan.

With these parameters, the procedure isolates the first word (or token) and returns to the calling routine. When returning to the calling routine, `START` and `LENGTH` are adjusted so that, on the next call to this procedure, the scan can continue to find the next word. Thus, successive calls to the procedure return words (or tokens) in sequence.

`LENGTH` is set to zero when the field `TEXT` has been completely scanned. The calling procedure tests for this.



ASL Quick Reference

ASL (Advanced Syntax Language) is a companion product to VISION:Builder, VISION:Inform, and VISION:Two.

Terminology

Term	Description
Procedure	A procedure defines an algorithm or sequence of calculations. It is composed of a series of procedure statements.
Procedure statement	A procedure statement begins on a new line and consists of an optional label followed by a command.
Label	A label identifies a specific statement. The label is optional. If used, place the label before a procedure statement and follow the label immediately (without intervening spaces) by a colon (:).
Command	A command identifies the kind of procedure statement. You can follow a command by keywords, functions, constants, names, qualifiers, expressions, and comments.
Function	A function is a subroutine that derives a value from other data values.
Keyword	A keyword identifies how certain parameters are used. Most keywords are optional.
Keyword operand (or just operand)	A keyword operand identifies the data values associated with a keyword. In the procedure statement, follow a keyword with one or more spaces and a keyword operand.
Keyword phrase	A keyword phrase is a keyword plus its operand.

References

For information on IBM Language Environment® Callable Services (CEExxx), refer to *IBM Language Environment for MVS and VM Programming Reference*.

Constants

There are six types of constants: character, integer, decimal, floating point, time, and pattern. A description of each follows:

- Delimit character constants by single quotation marks. Specify a single quotation mark within a constant by two consecutive single quotation marks. For example: 'A' or 'CAN''T'
- Specify integer constants with numeric digits only. If the integer constant is negative, place a minus sign (-) before the constant. If the integer constant is positive, you can place an optional plus sign (+) before the constant. For example: -2, 0, or +7
- Specify decimal constants with numeric digits, an optional sign, and a decimal point. If you use a plus sign (+) or a minus sign (-), make it the first character of the constant. For example: -2000.00, 3.99, or +7.25
- Specify floating point constants with an optional sign preceding an integer or decimal constant followed by a power of ten expressed in exponential notation. For example: 1.50E10 or 13.75E-5
- Specify a time constant, HH:MM:SS.nn...n (hours, minutes, seconds, decimal seconds) by the letter T, followed by a single quotation mark, the time constant, and a closing single quotation mark. For example: T'12:01:00.125'
- Specify a pattern starting with the letter P, followed by a single quotation mark, a string of special pattern symbols, and a closing single quotation mark. For example: P'##(999#)#999#-#9999' or P'ZZZ999'

Names

- Begin field names and statement labels with an alphabetic character (A-Z).
- Make the remaining seven characters either alphabetic characters (A-Z), numeric digits (0-9), or underscores (_).
- Enclose field names that do not conform to this syntax in double quotation marks.

Qualifiers

You can prefix a field name with a standard 1-character qualifier and a period. A qualifier identifies the type of field or file where the field exists. The following are valid qualifiers and their meanings:

Qualifier	Type or Location
Blank or N	New Master file
1-9	Coordinated files 1-9
T	Temporary field
F	Flag field
X	Transaction file
O or 0	Old master file
W	Working storage
V	Linkage section
A, B, E, H, J, K, M, Q, 1-9	Array (must match the qualifier that identifies the array)

Comments

Place comments anywhere following a semicolon (;), except on a continued line.

Arithmetic Expressions

Code arithmetic expressions with the operators +, -, *, and / for addition, subtraction, multiplication, and division, respectively.

For example: $A + B$ $A - B$ $B * C$ D / C

As in conventional algebraic notation, operations within an arithmetic expression are processed according to a specific hierarchy, from left to right. However, multiplication and division are performed prior to addition and subtraction unless this order is overridden by parentheses.

Logical Expressions

Use logical expressions in IF, CASE, and DO statements. Make logical expressions from conditional functions, relational expressions, or list expressions connected by logical operators.

- A conditional function is a function that returns a true or false condition.
- A relational expression is composed of two values connected by a logical operator.
In relational expressions, you can represent the logical operator as characters or as symbols:

EQ	or	=
NE	or	<>
GT	or	>
LT	or	<
GE	or	>=
LE	or	<=

- A list expression lists the specific values to test.
For example: NUMBER = 0001 0002 0003

Statement Syntax

The following describes the conventions used to provide a precise description of the syntax of a function or command. Enter commands and functions in the exact order given.

- Brackets [] indicate an optional parameter.
- Braces { } indicate a choice of entry. Unless a default is indicated, you must choose one of the entries.
- Required parameters do not have brackets or braces surrounding them.
- Items separated by a vertical bar (|) represent alternative items. Select only one of the items.
- An ellipsis (...) indicates that you can use a progressive sequence of the type immediately preceding the ellipsis. For example: name1, name2, and so on.
- Uppercase type indicates the characters to be entered. Enter such items exactly as illustrated. You can also use authorized abbreviations.
- Lowercase type specifies fields to be supplied by the user.
- Separate commands, keywords, and keyword phrases by blanks.
- Enter punctuation exactly as shown (parentheses, colons, and so on).

Continuation

You can write procedure statements on multiple lines. Terminate each line by a blank space followed by a comma. Continue the remainder of the statement on the following lines. For example:

```
IF      NAME      = 'THE ABC COMPANY ' ,
      AND NUMBER = '00001' ,
      OR   NUMBER = '0002' ,
      OR   NUMBER = '0003' ,
THEN
```

Built-In Functions

Specify functions by entering a function name followed immediately (with no intervening spaces) by a left parenthesis, one or more keyword phrases, and terminating with a right parenthesis.

Conditional Functions

Conditional functions return a true or false condition.

```
FIND( [ SEGMENT ] segment-name [ FIRST | LAST |
      NEXT | WHERE selection-expression ] )

LOCATE( [ ARRAY ] array-identifier { [ ROW row-number ]
      [ COLUMN column-number ] } )

SCAN( [ FIELD ] field-name [ FOR ] search-value/pattern
      { FROM LEFT | FROM RIGHT } [ NOTEQUAL ] )

VALIDATE( [ FIELD ] field-name { PATTERN
      P'pattern' | DATE } )
```

Value Functions

Value functions return an actual value, either a result from a table or part of an existing field.

```
LOOKUP( [ TABLE ] table-name [ ARGUMENT ] lookup-argument
      [ NEAREST | SMALLER | LARGER | INTERPOLATE ] )

PF( [ FIELD ] field-name [ START ] start-position
      [ LENGTH partial-length ] )
```

Commands

Long commands that have abbreviations are shown in braces { } to indicate choice. For example: { COMBINE | COM }, { CONTINUE | CONT }, { FIELD | FLD }, and so on.

```
ARRAY [QUALIFIER] qualifier-char,
      NAME definition-name,
      [OVERDEFINES qualifier-char]
```

```
AVERAGE [ITEM] summary-field ...,
        {BY group-field | AT LEVEL level-number}
```

```
CALL { [ PROCEDURE ] procedure-name |
      MODULE module-name |
      REPORT report-name |
      SUBFILE subfile-name |
      MODULE 'module-name' |
      CEEDATE | CEEDATM | CEEDAYS |
      CEEDYWK | CEEGMT | CEEGMT0 |
      CEEISEC | CEELOCT | CEEQCEN |
      CEEScen | CEESECI | CEESECS | CEEUTC } IBM Language
      [ USING parm ... ] } Environment (LE)
                           Callable Services
                           Routines (CEExxxx)
```

```
CASE [ WHERE ] logical-expression
```

```
CATALOG {SAVE [GROUP group-name] [REQUESTS request-name ...] |
        INSERT REQUEST request-name INTO group-name [AFTER request-name] |
        DELETE [GROUP group-name] [REQUESTS request-name ...] |
        REPLACE REQUESTS request-name ... |
        DUMP {ALL | ITEMS name ... } |
        LIST}
```

```
CHECKPOINT [COUNT number],
          [FILE dname],
          [TIME number {MINUTES | SECONDS}],
          [EOV dname],
          [OPERATOR],
          [PREFIX id-prefix],
          [COMMITONLY],
          [ALTERNATING]
```

```
COLLATE { REPORTS report-name ...
        [ KEYLENGTH length ] |
        ALL KEYLENGTH length }

{ COMBINE | COM } [ FIELDS ] field1 ... STORE result-field
  [ [ BLANKS ] number ]

COMPUTE STEMPnn = operand1 operator operand2,

        [ PICTURE P'picture' ],
        [ DECIMALS decimal-places ],
        [ ROUNDED ],
        AT LEVEL level-number

{ CONTINUE | CONT }

CONTROL [ NAME run-name ],

        [ DELIMITER 'x' ],
        [ { SCANONLY | SAMPLE | MAPDECODE } ],
        [ { TERM | CONTINUE } ],
        [ SORT { INTERNAL | EXTERNAL | NONE }, [ SUMMARIZE ] ],
        [ { NOLIST | NOSOURCE | LISTGEN } ],
        [ AMODE { 31 | 24 } ],
        [ ABEND ],
        [ GRANDSUM ],
        [ CORDONLY ],
        [ GETMAIN nnnnK ],
        [ SORTSIZE nnnnK ],
        [ REPTSIZE nnnnK ],
        [ FREESIZE nnnnK ],
        [ DB2 subsystem-id plan-name ],
        [ SQLID authorization-id ],
        [ EXPLAIN query-number ],
        [ { SYSDATE mmdyy | SYSDATE4 yyyyymmdd } ],
        [ DECMSG { YES | NO } ],
        [ PROMSG { YES | NO } ],
        [ RPTMSG { YES | NO } ]

COPY [ DDNAME ] ddname [ (member-name) ],

        [ FIXED ]

COUNT [ ITEM ] summary-field ...,

        { BY group-field | AT LEVEL level-number }
```

```

CUMULATE [ITEM] summary-field ...,
          {BY group-field | AT LEVEL level-number}

DATA {'text' | LITERAL('text', repeat-count) | field-name |,
      summary-function(field-name [level]) |,
      STEMPnn | COL(column-number) | SPACES(number-spaces)}...,
      AT LEVEL level-number

DEBUG [CLEAR],
      [DUMP],
      [LONGNAMES],
      [TRACE COMPCODE + IMSCALLA + MAPPING + SQLCALL]

DO      { [ WHILE logical-expression ]
          [ UNTIL logical-expression ]
          [ FORALL segment-name ]
          [ FORALL CELLS IN ARRAY array-identifier ]
          [ FORALL COLUMNS IN ARRAY array-identifier
            [ WITHIN ROW row-number ] ]
          [ FORALL ROWS IN ARRAY array-identifier
            [ WITHIN COLUMN column-number ] ]
          [ FOR integer ] } |
          [ CASE ]

DOCUMENT [CONVMSG],
          [XREF],
          [EXEctrace],
          [MAXLINES number]

ELSE

END [{DO | IF | PROC | REPORT | SECTION}]

```

```
[extract-name:] EXTRACT DBDNAME dbdname,

    ENTIRE qual-char,
    [{DLI | HDAM}],
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [AUTODEF],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ |
          TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
          EOF | EOFPLUS}],
    [INFO 'text']
```

```
[extract-name:] EXTRACT DDNAME ddname,

    {[ITEMS] field-name ... | ENTIRE qual-char},
    [KEYS key-field ...],
    [VARCHARMAX NOPREFIX varchar-field1 ...],
    [VARCHARMAX WITHPREFIX varchar-field2 ...],
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [ABBREVIATED],
    [RECFM {FIXED | VARIABLE | UNDEFINED |
          KEYVSAM | ENTRYVSAM | PACKED}],
    [BLKSIZE block-size],
    [AUTODEF],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ |
          TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
          EOF | EOFPLUS}],
    [INFO 'text']
```

```
[extract-name:] EXTRACT FILE subfile-name,

    {[ITEMS] field-name ... | ENTIRE qual-char},
    [KEYS key-field ...],
    [VARCHARMAX NOPREFIX varchar-field1 ...],
    [VARCHARMAX WITHPREFIX varchar-field2 ...],
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [ABBREVIATED],
    [RECFM {FIXED | VARIABLE | UNDEFINED | KEYVSAM |
          ENTRYVSAM | DLI | HDAM | DB2 | PACKED}],
    [BLKSIZE block-size],
    [AUTODEF],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ |
          TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
          EOF | EOFPLUS}],
    [INFO 'text']
```

```
[extract-name:] EXTRACT TABLE "authid.tablename",
    {CREATE | DELETE | INSERT | DROP},
    [TABLESPACE tablespace-name],
    [DATABASE database-name],
    ITEMS {field-name | PF(field-name start length) ...},
    [KEYS {key-field | PF(key-field start length) ...}],
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [ABBREVIATED],
    [AUTODEF],
    [DEFNAME definition-name],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ |
          TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
          EOF | EOFPLUS}],
    [INFO 'text']

field name: { FIELD | FLD } [ TYPE ] field-type [ [ LENGTH ] field-length ]
    [DECIMALS decimal-places ]
    [FLOAT floating-edit-char ]
    [FILL fill-edit-char ]
    [TRAIL trailing-edit-char ]
    [EDLEN edit-length ]
    [INIT initial-value ]
    [HEADING 'line1' [ 'line2' ] ]

FILE AUDIT [DDNAME ddname]
```

```
FILE CORDn {[NAME] definition-name | SQL "sql-select-statement"},
    [MOSAIC],
    [CHKORDER],
    [ARRAY],
    [{SEGMENT segment-name WHERE "sql-where-clause" ... |
      SEGMENT segment-name SSA "pre-selection-ssa" ... }],
    [{PASSWORD password | AUTHID authorization-id}],
    [IOPLUGIN module-name],
    [DDNAME ddname]
```

Additional keywords for Indexed Direct Coordination (ICF):

```
[DIRECT BY q.fldname]
```

Additional keywords for Standard Coordination:

```
[STANDARD],
[ALLRECS | MATCHONLY],
[KEYNAME field-name ...]
```

Additional keywords for Chained Coordination:

```
[CHAIN TO qualifier],
[KEYNAME field-name ...]
```

Additional keywords for User Read files:

```
[USERREAD],
[GENERIC field-name]
```

```
FILE MASTER {INPUT | UPDATE},
    {[NAME] definition-name | SQL "sql-select-statement"},
    [ACCESS {SEQUENTIAL | DIRECT | PHYSICAL}],
    [KEYS {UNIQUE | EQUAL | NONE}],
    [STARTKEY 'key-value'],
    [ENDKEY 'key-value'],
    [KEYNAME field-name ...],
    [MOSAIC],
    [CHKORDER],
    [ONEBUFFER],
    [{SEGMENT segment-name WHERE "sql-where-clause" ... |
      SEGMENT segment-name SSA "pre-selection-ssa" ... }],
    [{PASSWORD password | AUTHID authorization-id}],
    [IOPLUGIN module-name],
    [DDNAME ddname]
```

```
FILE REJECT [DDNAME ddname]
```

```
FILE REPn [NAME] report-file-name,  
        [DDNAME name]
```

```
FILE REPORT [DDNAME ddname]
```

```
FILE SUBFn [NAME] subfile-name,  
        [TABLE "authid.tablename" {CREATE | DELETE | INSERT | DROP}],  
        [TABLESPACE tablespace-name],  
        [DATABASE database-name],  
        [DELETEDSEGS],  
        [AUTODEF],  
        [{PASSWORD password},  
        [IOPLUGIN module-name],  
        [DDNAME ddname]
```

```
FILE TRAN [[NAME] definition-name],  
        [GROUPS group-name ...],  
        [CHKORDER],  
        [PASSWORD password],  
        [IOPLUGIN module-name],  
        [DDNAME ddname]
```

```

FORMAT [HEIGHT number],
    [WIDTH number],
    [DATEPOS {UL | UR | UM | LL | LR | LM | NO}],
    [PAGEPOS {UL | UR | UM | LL | LR | LM | NO}],
    [TITLEPOS {TOP | BOTTOM}],
    [HEADINGS {YES | NO | NAME}],
    [HEADPOS {ABOVE | BELOW}],
    [DATEFMT {DATE | TODAY | TODAYX | ISDATE | JULIAN | JULANX |
    mmddyy}],
    [BORDER {YES, NO, 'x'}],
    [STARTPAGE {number | PAGE}],
    [MAXPAGES number],
    [LINESPERPAGE number],
    [DETAILSPACING number],
    [INCOMPLETESUM 'x'],
    [NODATA {SKELETON | NOREPORT}],
    [SUBTITLE {REPEAT | NOREPEAT | NEWPAGE}],
    [SUMMARYLABELS {SPACE | NOSPACE | SUPPRESS}],
    [LINENUMS {NONE | LEFT | RIGHT | BOTH}],
    [IMAGES number [IMGTITLE {LOGPAGE | PHYPAGE | NEWPAGE}]]
    [METHOD {STDLIST | ALTLIST | CSV | TAB | HTML | PLAINTEXT |
    RAWDATA | CLEARACCESS}],
    [STYLE number],
    [DDNAME ddname],
    [SUMFILE ddname],
    [AUTODEF]

GO [ TO ] jump-to-label

GROUP [BY] field-name ...
    [AT LEVEL level-number],
    [SUBTITLE [NEWPAGE]],
    [LABEL field-name]

IF [ CONDITION ] logical-expression [ THEN ]

INCLUDE [ITEM] item-name,
    [DATEFMT {DATE | TODAY | TODAYX | ISDATE | JULIAN | JULANX | mmddyy}],
    [INFO 'text']

ITEM [FIELD] field-name ...,

```

```

[SPACES number],
[PICTURE P'pattern'],
[ENDLINE],
[NONPRINT],
[VWIDTH],
[NOWRAP],
[SPLITOK],
[CSVEDIT {QUOTE | TRUNCATE [DECIMALS number]}]

```

LEAVE

```

LET      [ FIELD ] result-field = source-expression
[ WITH ] [ EDIT P'pattern' ] [ ROUNDING ]
[ JUSTIFY { LEFT | RIGHT } ]

```

```

LINE [ {'text' | LIT[ERAL]('text', repeat-count) | field-name |,
      summary-function(field-name [level]) |,
      STEMPnn | COL(col-num) | SPACES(num-spaces)}...] [AT LEVEL level]

```

```

LINKAGE [AREA] number,
        NAME definition-name

```

```

LISTCNTL [ALTLIST {YES | NO}],
        [FILESUM {YES | NO}],
        [INDEF {YES | NO}],
        [INGLOSS {YES | NO}],
        [INREQ {YES | NO}],
        [CATREQ {YES | NO}],
        [MAPREQ {YES | NO}],
        [SQLSTAT {YES | NO}],
        [MOSAICSTAT {YES | NO}]

```

```

LISTLIB GLOSSARY {ARRAY | FILE | GROUP | TABLE | VIEW | INVIEW},
        {ALL | [ITEMS] name ...}

```

```

LISTLIB NAMES {ALL | ARRAY | FILE | GROUP | TABLE | VIEW}

```

```

{ LOCATE | LOC } [ ARRAY ] array-identifier
  { [ ROW row-number ]
    [ COLUMN column-number ] }

```

```
MAX [ITEM] summary-field ...,
    {BY group-field | AT LEVEL level-number}

MIN [ITEM] summary-field ...,
    {BY group-field | AT LEVEL level-number}

MULTILIB {OFF | ORDER ddname ...}

NEWPAGE [AT LEVEL] level-number

ORDER [BY] field-name ...,
    [DESC field-name ...]

OVERRIDE [DDNAME] old-ddname,
    WITH new-ddname

OWNCODE [MODULE] module-name,
    HOOKS hook-number ...

PERCENT [ITEM] summary-field ... OF denominator-field,
    {BY group-field | AT LEVEL level-number},
    [DECIMALS decimal-places]

PREFACE [LINE] 'text' ...

[proc-name:] PROC [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ |
                        TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
                        EOF | EOFPLUS}],
```

```
[TEMPREINIT],
[SELECTIONCONTROL {YES | NO}],
[MAXITEMS number],
[INFO 'text'],
[PARALLEL_LOOPING] (use with caution)
```

```
RATIO [ITEM] summary-field ..., TO denominator-field,
    {BY group-field | AT LEVEL level-number},
    [DECIMALS decimal-places]
```

```
{ RELEASE | REL } { [ SEGMENT ] segment-name | ARRAY array-identifier }
```

```
{ REPLACE | REP } [ STRING ] search-string [ IN ] modify-field
    [ WITH ] substitute-value
```

```
[report-name:]REPORT [ITEMS] {field-name | report-function} ...,
    [SUMMARYONLY],
    [SINGLESPEACE],
    [GRANDSUMS],
    [EMPTYFIELD {INCLUDE | EXCLUDE}],
    [SELECTIONCONTROL {YES | NO}],
    [MAXITEMS number],
    [ABBREVIATED],
    [FILE report-file-name],
    [TYPE {NORMAL | SUBROUTINE | INIT | PRE MASTER READ |
        TYPE1 | TYPE2 | TYPEM | TYPE3 | TYPE4 |
        EOF | EOFPLUS}],
    [INFO 'text']
```

```
RETRIEVE {ARRAY | FILE | GROUP | REQUEST | TABLE | VIEW | EOF},
    {ALL | [ITEMS] name ...},
    [NEWNAME name]
```

```
{ RETURN | RET }
```

```
ROUTE {[REPORTS] report-name ... | ALL},
    [KEYVALUE 'data-value'],
    TO destination-names ...,
    [DEFER]
```

```
SECTION {PAGETITLE | COLUMNHEADING | SUMMARY}
```

SIZE *number-of-lines* LINES

SKIP *number-of-lines* LINES [AT LEVEL *level-number*]

TITLE [LINE] '*text*' ...

TOTAL [ITEM] *summary-field* ...,
 {BY *group-field* | AT LEVEL *level-number*}

TRACK [NAME] *item-name*,
 [GENERIC],
 [TYPE {FILE | ARRAY | TABLE | TRAN | REQUEST | REQGROUP}],
 [FILENAME *file-name*],
 [*{*EXPIRE *mmddy* | RETAIN *days**}*],
 [USERID *userid*]

TRANSFER [TO] { NEXT_MASTER | TYPE_1 | TYPE_2 }

WORK [AREA] *number*,
 NAME *definition-name*

XREP [LINE] '*text*'

Relationship of ASL Statements to Fixed-Format-Syntax Statements

This appendix relates the procedural ASL functions and commands to their approximate VISION:Builder statements. They are listed in the order in which they appear in the body of this booklet, grouped by function, then by command.

ASL statements frequently generate more than one VISION:Builder statement. The relationships provided in this appendix are only approximate and are provided for the benefit of those users familiar with fixed format VISION:Builder statements.

Note: Functions and commands pertaining to arrays are applicable to *VISION:Builder* and *VISION:Two*, but not *VISION:Inform*.

ASL Function	VISION:Builder Operation
FIND	FS when a lower level segment is named. RD, RE, or RG when a root segment from a request read coordinated file is named.
LOOKUP	TL, TN, TS, TB, or TI, depending on the optional keyword used.
PF	A partial field operation is done on the field named in the function. The PF function can be used on as many fields in the command as needed.
SCAN	SL, SR, or SN, depending on the keywords used in the function.
VALIDATE	CV or DV, depending on the keywords used in the function.
CALL	Branches to a subroutine request or procedure, a report, or a subfile. An external program can also be called with this command.
CASE	Creates a logical expression showing what is to be performed when the expression is true.
COMBINE	Cn operator is used with a keyword phrase setting up the number of blanks between fields.
CONTINUE	Performs a GO END procedure.

ASL Command VISION:Builder Operation	
DO	Sets up a loop depending on the keyword phrases used on the command. The keywords UNTIL and WHERE set up back branch loops with the FOR keyword putting a maximum number of back branches to perform. The FORALL keyword sets up an automatic loop for a lower level segment. The CASE keyword sets up a case block that does not establish any loop.
ELSE	Starts the block of code to be performed when the logical expression on the preceding IF command fails or all of the CASE commands within a DO CASE block fail.
END	Delimits the end of a DO block or an IF-THEN-ELSE set of code.
FIELD	TF statement for a temporary field.
GO	Performs a forward branch to a PR statement.
IF	Creates a logical expression showing what is to be performed when the expression is true. Optionally, an ELSE statement can be entered before the corresponding END statement to allow processing of code for a false condition.
LEAVE	Can only be coded in a DO block without the CASE keyword. The DO block is generated as a subroutine, and the LEAVE command in a DO block generates a GO RETURN command.
LET	An R, JR, or JL operation is performed, depending on the keywords at the end of the LET command.
LOCATE	An LR, LC, or LD, depending on the keyword phrases added after the array qualifier is named.
RELEASE	An RS or LA, depending on the keyword phrase provided on the release command.
REPLACE	An SS operator.
RETURN	A GO RETURN from a subroutine procedure.
TRANSFER	A GO NEXT MASTER, GO TYPE 1, or GO TYPE 2.

This appendix contains additional information about topics described in this book. In some instances, more detail is necessary about the entries that are valid for an operand or more background information might be helpful in understanding a part of the syntax used (for example, qualifiers).

Reserved Words

Do not make field names the same as system defined names (reserved words) such as commands, functions, keywords, or operators. Making field names the same as system defined names could produce errors.

If you need to, you can use any of these reserved words as a field name if you surround the name by double quotation marks or enter all optional keywords in the statement.

The following is a list of restricted ASL words.

ARG	FDNAME	LEAVE
ARGUMENT	FIELD	LEN
ARR	FIELDS	LENGTH
ARRAY	FILE	LET
ASTATUS	FILEID	LEVEL
ATTNID	FILL	LN
AVG	FIND	LNUMBER
BLANKS	FIRST	LOC
CALL	FLD	LOCATE
CASE	FLDS	LOOKUP
CHAR	FLOAT	LOWVALU
CHKP	FLT	LR
CNT	FOR	LRGR
Cn	FORALL	LS
COL	FROM	LSTART
COLLATE	FS	LSTATUS
COLUMN	GE	LT
COM	GO	LU
COMBINE	GOSC	MAX
COMMAND	GOTO	MIN
COND	GS	MISSPASS
CONDCODE	GT	MN
CONDITION	HEAD	MNUMBER
CONT	HEADING	MOD
CONTINUE	HEX	MODE
COUNT	HIGHVALU	MODULE
CSTATUS	IF	MS
CUM	IN	MSG
CV	INIT	MSGLINE
DATE	INT	MSTART
DEC	INTERPOLATE	MSTATUS
DECIMALS	ISDATE	NE
DELETE	JL	NEAREST
DLI	JR	NEXT
DO	JST	NEXT_PROC
DV	JULIAN	NEXTPROC
ECORD	JUSTIFY	NOTEQUAL
EDIT	KEY	NRST
EDLEN	LA	NS
ELSE	LAB	OPERID
END	LABEL	OPERL
EOF	LARGER	OUTPUT
EQ	LAST	OWN
ERROR	LC	
EXPR	LD	
EXPRESSION	LE	

Figure C-1 Reserved Words (Page 1 of 2)

PAGE	RG	TABLE
PASSWORD	RN	TB
PAT	RNUMBER	TERMID
PATTERN	ROUNDING	TEXT
PCT	ROUTE	THEN
PF	ROW	TI
PLI	RS	TIME
PN	RSTART	TL
PNUMBER	RSTATUS	TN
PROC	RTO	TO
PROCEDURE	SCAN	TODAY
PS	SEG	TOT
PSTART	SEGNAME	TOTAL
R	SET	TRACE
RATIO	SL	TRAIL
RD	SMALLER	TRAN
RE	SMLR	TRANCODE
READ	SN	TRANSFER
REL	SR	TRL
RELEASE	SS	TS
REP	SSCOUNT	TYPEUNTIL
REPLACE	START	USING
REPORT	STORE	VAL
REQ	STR	VALIDATE
REQUEST	STRAN	WHERE
RESTART	STRING	WHILE
RET	SUB	WITH
RETURN	SUBFILE	XTRAN
RETURNCD	TAB	

Figure C-1 Reserved Words (Page 2 of 2)

Qualifiers

You can use a standard 1-character qualifier and a period to prefix a field name. Qualifiers identify the type of field or file where the field exists. The following are valid qualifiers and their meanings:

VISION:Builder		VISION:Inform	
Qualifier	Type or Location	Qualifier	Type or Location
Blank or N	New master file.	Blank or N	Primary file.
O or 0	Old master file.	O or 0	Primary file.
1-9	Coordinated files 1-9.	1-9	Synchronized files 1-9.
T	Temporary field.	T	Temporary field.
F	Flag field.	F	System field.
X	Transaction file.		
W	Working storage.		
V	Linkage section.		
A, B, E, G, H, J, K, M, Q, 1-9	Array (must match the qualifier that identifies the array).		

Patterns

There are two pattern types for VISION:Builder and VISION:Inform applications: validation patterns and edit patterns. This section discusses each pattern type.

Validation Patterns

Use validation patterns in SCAN, VALIDATE, and REPLACE statements. Enclose patterns in single quotation marks, preceded by a P. You can make patterns up to 30 characters long. Code one pattern symbol for each character of the field being validated.

Note: A minus (-) sign before any of these characters means scanning for other than the specified pattern. Minus C (-C) is not a legal entry. Also, you must use system delimiters to surround literals in a scan pattern.

You can use the following validation pattern symbols:

Symbol	Meaning
Z	Alpha (A-Z).
z	Alpha (A-Z, a-z).
A	Alpha (A-Z) or blank.
a	Alpha (A-Z, a-z) or blank.
D	System delimiter.
9	Numeric (0-9).
I	Numeric (0-9) or blank.
Y	Alpha (A-Z) or numeric (0-9).
y	Alpha (A-Z, a-z) or numeric (0-9).
X	Alpha (A-Z), numeric (0-9), or blank.
x	Alpha (A-Z, a-z), numeric (0-9), or blank.
B	Blank.
C	No validation.
Literal(s)	Any character(s).
User-defined	Provided by you at system installation time.

Edit Patterns

Create edit patterns to edit the result field in the LET statement. Using edit patterns, you can specify whether to store additional characters into the result field or truncate existing ones. You can make the edit pattern up to 30 characters long.

There are two styles of patterns available: the delimiter style or the COBOL style.

- Delimiter style patterns use the delimiter character as the character/digit selector symbol, the decimal point as a decimal alignment symbol, and all other characters as literal insertion characters. Use this edit pattern style when editing character data or when editing numeric data.
- COBOL style patterns use the symbols described in [Numeric Data \(Packed, Zoned, and Fixed Point Binary Only\)](#). Use this edit pattern style only with numeric data.

Character String Data

Enter the edit pattern as a picture, with the delimiter (#) as an edit pattern symbol. If required, you can insert additional edit symbols. For example, you can specify a pattern as ###/##/##.

- The first character in the edit pattern represents the character on the extreme left.
- Truncate characters on the right by not entering delimiters to represent them.
- If left truncation or extraction of characters in the middle of a field is preferred, use partial fielding to specify the partial field start position and number of characters to be stored.

Numeric Data (Packed, Zoned, and Fixed Point Binary Only)

Use the edit pattern to truncate digits following the decimal point or non-significant zeroes to the left of the decimal point. The pattern causes digits surrounding the decimal point to be stored.

The period (.) has special meaning in the edit pattern.

- The first period is interpreted as a printable decimal point and is used to align decimal points.
- If no period exists, only the integer portion of a number is stored.

Leading zeroes are suppressed if they are the leading characters of the picture. When a negative value is output, the sign is stored to the left of the first digit or replaces the first leading zero.

You can use the following edit pattern symbols for numeric data:

Symbol	Meaning
9	Digit select. Represents a character position that contains a numeral.
Z	Digit select. Same as 9, except when the result contains a zero as a leading character, the zero is replaced by a space (blank). Signs do not print.
*	Check protection. Same as Z, except a zero in a leading position is replaced by an asterisk (*).
.	Decimal position. Represents a decimal point for alignment in the edit picture. For the output, the decimal point represents a position into which a character is inserted.
,	Grouping character. Represents an output position into which a grouping character is placed.
+ -	Sign control. These symbols are used as editing sign control symbols to position the edit sign.
\$	Currency symbol. Represents an output position into which the currency symbol is placed.

Rules for Edit Patterns

- You can edit source digits to the output positions with or without suppression of leading zeroes or check protection (for example, 99.9 = no suppression, ZZ.9 = suppress leading zeroes to decimal point, *.* = check protects the entire field).
- You can specify a leading or trailing sign. For example, +99, -99, 99-, 99+.

Symbol in Edit Picture	Result if Positive Data Item (Includes Zero)	Result if Negative Data Item
+	+	-
-	SPACE	-

- You can specify a leading sign as fixed (leftmost position on output) or floating. For example, +99.9 = fixed, +++.9 = floating.
- You can specify a currency symbol as fixed (leftmost position on output) or as floating (for example, \$99.9 = fixed, \$\$\$\$.9 = floating). The currency symbol and the sign symbols are mutually exclusive as fixed insertion characters in a given picture.

- Indicate floating insertion of leading sign or currency symbol by a string of at least two occurrences of the same characters (\$+–) to represent the leftmost numeric position in the output. For example, +99.9 = fixed, +++.9 = floating.
 - The currency symbol and the sign symbols are mutually exclusive as floating insertion characters in a given picture.
 - A fixed insertion character can also be specified in the same picture with a floating insertion character as long as it is a different character.
- Use floating insertion of leading sign or currency symbols to suppresses leading zeroes in the same manner as the digit select character Z.
- Suppression symbols (Z\$*+–) can occur in an edit picture as follows:
 - Any or all of the leading numeric positions to the left of the decimal point are represented by suppression symbols (for example, \$\$,\$\$\$99).
 - All of the numeric positions in the edit picture are represented by suppression symbols (for example, \$\$,\$\$\$,\$\$).

If the suppression symbols appear to the left of the decimal point, any leading zero in the data that appears in a character position corresponding to a suppression symbol in the picture is replaced by the space character or check protection character.

Suppression terminates at the first non-zero digit in the data or at the decimal point, whichever comes first. For example, \$\$\$99 with data 001.23 produces \$1.23.

- If all numeric positions in the picture are represented by suppression symbols and the value of the data is not zero, the result is the same as if the suppression characters were to the left of the decimal point.
- If the value of the data is zero, the entire output becomes spaces if all numeric positions in the picture are represented by suppression symbols. For example, \$.\$\$ with data 0.00 produces all spaces.
- If check protection is used, all positions become the protection character, except for a decimal character. For example, *.* with 0.00 produces *.*.

Any grouping characters embedded in the string of suppression characters or to the immediate right of the string are suppressed if the left adjacent numeric position is suppressed. For example, \$\$,\$\$\$99+ with data 0012.34 produces \$12.34+.

- Using a suppression symbol throughout a field suppresses a trailing sign on all zero fields (for example, ZZ.ZZ+ with value +00.01 produces .01+, while +00.00 produces all spaces). Delimiters should not be used with Z picture edit as ZZZ,ZZZ.##, but rather, as ZZZ,ZZZ.99.
- The grouping character can occur on either side of the decimal position character (for example, \$\$,\$\$\$999,99 international metric standard).

- Truncating the significant trailing digits is valid (for example, 99.99 with data 12.345 produces 12.34). Truncating the significant leading digits is not valid. For example, 9.99 with data 12.34 produces + or the “uneditable” VISION:Builder symbol.
- A trailing character, if specified, cannot be the same symbol used as a floating or leading character in a given pattern. For example, +99.9+ is invalid; +99.9- is valid.

Output Edit

Use the output edit entries (floating, filling, or trailing) only with numeric fields (packed, zoned, and fixed point binary). These entries edit the data before reporting it.

Commas

Commas print in these fields where they are preceded by a significant digit. To suppress printing of commas, use edit suppress character ‘Z’ or the Override “Picture” Edit on the reporting statement.

Standard Notation

Do not make an entry for floating point numbers. They always print in scientific notation. The output edit length must exceed six to include two signs, a decimal point, E, and two exponent digits. Standard notation for floating point numbers, where Xs represent the fraction and Ys the exponent, is:

± .XXXXXXXXE ±YY

Floating/Edit Suppress

An entry other than Z “floats” the value of the entry and prints to the immediate left of the first significant digit in the printed report.

- If this is not used, a leading blank prints if positive, a minus if negative.
- If this operand contains a Z, all commas, leading zeroes to the left of the decimal point, and the decimal point, if specified, are suppressed.

Float (Floating-Edit-Char)

This operand is valid only for packed, zoned, and fixed point binary field types.

Code	Result
Blank	When no output codes are specified, a leading blank prints if the value of the field is positive; a leading minus sign prints if the value of the field is negative; commas print and the decimal point prints if the decimal places are specified. A zero value, in a field where a decimal place is specified, prints as a decimal point followed by as many zeroes as there are decimal places.
\$	A floating dollar sign prints before the first value when a control break occurs and when summaries are taken.
+	A leading + sign prints if the value of the field is positive, – if negative.
–	A leading – sign prints if the value of the field is negative (default specification).
Z	The printing of leading zeroes is suppressed from the left to the first non-zero digit or decimal place. Negative signs print, but no space is allocated for them.
Any other character	A floating leading character prints on control breaks and on summaries.

Fill (Fill-Edit-Char)

The value of this operand prints in every position from the leftmost portion of the field until the first non-zero digit is encountered. This operand is valid only for packed, zoned, and fixed point binary fields.

Code	Result
Any character	Replaces leading zeroes.

Trail (Trailing-Edit-Char)

The value of this operand prints following positive and/or negative values. This operand is valid only for packed, zoned, and fixed point binary fields.

Code	Result
+	A trailing + sign prints if the value of the field is positive.
-	A trailing - sign prints if the value of the field is negative.
)	Encloses negative field values in parentheses. If no filling character is specified, the left parenthesis prints before the first significant digit or decimal point, whichever comes first. If a floating character and this character are specified, both can print. The floating character prints inside the parentheses: (\$43.50). Only a single floating sign is permissible with the trailing ")".
C	Prints a trailing "CR" for a negative value. Blanks follow a positive value.
D	Prints a trailing "DB" for a negative value. Blanks follow a positive value.
Any other character	Prints a trailing character for negative values only.

Edlen (Edit-Length)

The number of print positions for reporting the field.

Code	Result
Blank	For normal or fixed length fields, the system computes the length. For variable length fields, the system computes the length as the shorter of either the defined field length or report page width minus spaces before columns.
Any two-digit number	For normal or fixed length fields, the system uses the output edit length or field, including any floating, filling, and/or trailing characters. For variable length fields, the system uses the width of the column. Text automatically wraps until it is exhausted.

Valid Field Types and Default Field Lengths

The following table gives the valid entries for field type and field length and the defaults for field length.

FIELD TYPE	FIELD RANGE	DEFAULT FIELD LENGTH
C Character	1-255 Bytes	16
Z Zoned	1-15 Bytes	15
P Packed	1-15 Bytes	8
V Variable	1-999 or 1H-99H (H implies 00)	NULL
E Floating point binary	4-Bytes only	4
F Fixed point binary	1-4 Bytes	4
L Time HOUR:MIN:SEC	$\left[\frac{Nn+7}{2} \right]$ to 8	8
		8
		$\left[\frac{Nn+5}{2} \right]$ to 8
S Time MIN:SEC	Where Nn is the number of decimal-of-seconds digits. (Nn ≤ 9) and [] around the equation means use only the integer portion of the number.	
D Lilian Date	4 Bytes only	4

A flag is an internal indicator defined by VISION:Builder, VISION:Inform, and VISION:Two, which designates the existence of certain conditions during the application run. Special words function as flags. The names of the flags are not reserved; you can use flags as field names without restriction. You can also use flags in processing and reporting statements. The F qualifier identifies a flag to VISION:Builder.

Note: Flags pertaining to arrays are not applicable to *VISION:Inform*. Also, in this appendix, the symbol  or (*VISION:Builder* 4000 Model Series Only) designates that the feature or function being described only applies to the *VISION:Builder* 4000 model series and not to the *VISION:Two* 2000 model series.

The VISION:Builder software system has the following processing options, available in both the 4000 and 2000 model series.

- IMS™ — The IMS option provides support for processing information in IBM IMS Databases using the standard DL/I processing facilities.
In this appendix, the symbol  or (IMS Only) designates that the specification, feature, or function being described only applies to the IMS option.
- DB2 — The DB2 option provides support for processing information in IBM Database2 Relational Tables using the standard SQL processing facilities.
In this appendix, the symbol  or (DB2 Only) designates that the specification, feature, or function being described only applies to the DB2 option.
- GDBI — The GDBI option provides a facility for interfacing user code with the standard mechanisms of VISION:Builder to perform I/O operations using any database processing facilities.
In this appendix, the symbol  or (GDBI Only) designates that the specification, feature, or function being described only applies to the GDBI option.

Flag	Application	Format
ASTATUS	Indicates the status of an array operation.	4 bytes character
CHKP I	Directs VISION:Builder to take a checkpoint before reading the next master file root segment.	1 byte character
CKPTID I	Contains the checkpoint ID of the last checkpoint taken.	8 bytes character
COLUMN	Indicates the number of the current column of the array.	4 bytes fixed
COMMAND G	Contains the command identifier.	8 bytes character
CONDCODE	Modifies VISION:Builder condition codes (MVS, CMS), terminates job at end of job step (VSE). (User modifiable)	2 bytes binary
CSTATUS	Indicates that a CALL was suppressed by VISION:Builder and gives the reason.	4 bytes character
DATE	Records the operating system date in the format: MMM DD, YYYY.	12 bytes character
DELETE 4	Directs VISION:Builder to delete the current master file record, to delete all segment occurrences having empty key field values, or to reject the current transaction.	1 byte fixed
ECORD	Indicates the match condition between the master record key field and the key fields of the current coordinated records.	9 bytes character
EOF	Detects or forces end of file on any sequentially input files. (User modifiable)	11 bytes character
FDNAME G	Contains the file name of the mapped file.	8 bytes character
FILE G	Contains the VISION:Builder logical file name.	8 bytes character
FILEID G	Contains the file identification of the mapped file.	8 bytes character

Flag	Application	Format
ISDATE	Records the operating system date in the format: <ul style="list-style-type: none"> ■ YYYYMMDD (normal usage) ■ YYYY-MM-DD (formatted reports) 	8 bytes character 10 bytes character
JULANX	Records the operating system date in the format: <ul style="list-style-type: none"> ■ YYYYDDD (normal usage) ■ YYYY.DDD (formatted reports) 	7 bytes character 8 bytes character
JULIAN	Records the operating system date in the format: <ul style="list-style-type: none"> ■ YYDDD (normal usage) ■ YY.DDD (formatted reports) 	5 bytes character 6 bytes character
LILIAN	Date type field containing the Lilian date as the number of days since the beginning of the Gregorian calendar (October 14, 1582). The valid range of Lilian dates is 1 – 3,074,324 (October 15, 1582 to December 31, 9999). For example, a Lilian date with a value of 152384 converts to the standard date of December 31, 1999.	4 bytes integer (date type)
LNUMBER	Indicates the number of characters in the left part of the field just scanned. (User modifiable)	4 bytes fixed
LSTART	Indicates the starting location of the left part of the field just scanned. (User modifiable)	4 bytes fixed
LSTATUS	Indicates the status of segment operations.	4 bytes character
M4AUDIT ⁴	Provides the number of deleted master file records output to M4AUDIT during application execution.	4 bytes fixed
M4CORD1	Provides the number of records read from M4CORD1 during application execution.	4 bytes fixed
M4CORD2	Provides the number of records read from M4CORD2 during application execution.	4 bytes fixed
M4CORD3	Provides the number of records read from M4CORD3 during application execution.	4 bytes fixed

Flag	Application	Format
M4CORD4	Provides the number of records read from M4CORD4 during application execution.	4 bytes fixed
M4CORD5	Provides the number of records read from M4CORD5 during application execution.	4 bytes fixed
M4CORD6	Provides the number of records read from M4CORD6 during application execution.	4 bytes fixed
M4CORD7	Provides the number of records read from M4CORD7 during application execution.	4 bytes fixed
M4CORD8	Provides the number of records read from M4CORD8 during application execution.	4 bytes fixed
M4CORD9	Provides the number of records read from M4CORD9 during application execution.	4 bytes fixed
M4NEW ④	Provides the number of records output to M4NEW during application execution.	4 bytes fixed
M4OLD	Provides the number of records read from M4OLD during application execution.	4 bytes fixed
M4REJECT ④	Provides the number of records output to M4REJECT during application execution.	4 bytes fixed
M4SUBF0	Provides the number of records output to M4SUBF0 during application execution.	4 bytes fixed
M4SUBF1	Provides the number of records output to M4SUBF1 during application execution.	4 bytes fixed
M4SUBF2	Provides the number of records output to M4SUBF2 during application execution.	4 bytes fixed
M4SUBF3	Provides the number of records output to M4SUBF3 during application execution.	4 bytes fixed
M4SUBF4	Provides the number of records output to M4SUBF4 during application execution.	4 bytes fixed
M4SUBF5	Provides the number of records output to M4SUBF5 during application execution.	4 bytes fixed
M4SUBF6	Provides the number of records output to M4SUBF6 during application execution.	4 bytes fixed
M4SUBF7	Provides the number of records output to M4SUBF7 during application execution.	4 bytes fixed
M4SUBF8	Provides the number of records output to M4SUBF8 during application execution.	4 bytes fixed

Flag	Application	Format
M4SUBF9	Provides the number of records output to M4SUBF9 during application execution.	4 bytes fixed
M4TRAN ⁴	Provides the number of records read from M4TRAN during application execution.	4 bytes fixed
MISSPASS	Indicates the status of the follow-up pass in sequential coordination for master file records.	1 byte character
MNUMBER	Indicates the number of characters in the middle part of a field just scanned. (User modifiable)	4 bytes fixed
MODE ^G	Contains information about the application mode of operation.	2 bytes character
MSTART	Indicates the starting location of the middle part of a field just scanned. (User modifiable)	4 bytes fixed
MSTATUS ^G	Used by the mapping request to instruct VISION:Builder on a specific action to take subsequent to completion of the mapping request. (User modifiable)	6 bytes character
OWN	Provides a means of communication between user routines and own-code exits. (User modifiable)	16 bytes character
PAGE	Indicates placement of page numbers in formatted reporting.	6 bytes character
PASSWORD ^G	Contains the password from the RF statement.	8 bytes character
RESTART ^T	Indicates restart ID or blank if not a restart.	8 bytes character
RETURNCD	Determines the results of the CALL as set by the generalized system interface CALL routine.	4 bytes fixed
RNUMBER	Indicates the number of characters in the right part of a field just scanned. (User modifiable)	4 bytes fixed
ROW	Indicates the row number in the array.	4 bytes fixed
RSTART	Indicates the starting location of the right part of the field just scanned. (User modifiable)	4 bytes fixed
RSTATUS	Indicates the results of a read operation.	4 bytes character
SEGNAME ^G	Contains the segment name as defined on the LS statement.	8 bytes character

Flag	Application	Format
SQL [Ⓛ]	Indicates the status of inserting a row into an SQL table.	4 bytes character
SSCOUNT	Counts the number of matches found during a REPLACE operation.	2 bytes fixed
STRAN [Ⓛ]	Interrogates status of transactions at each segment and indicates which have been applied to the master file.	9 bytes character
TIME	Records the time of day a job was started in the format: HH.MM.SS (hours, minutes, seconds).	8 bytes character
TODAY	Records the operating system date in the format: <ul style="list-style-type: none"> ■ MMDDYY (normal usage) ■ MM/DD/YY (formatted reports) 	6 bytes character 8 bytes character
TODAYX	Records the operating system date in the format: <ul style="list-style-type: none"> ■ MMDDYYYY (normal usage) ■ MM/DD/YYYY (formatted reports) 	8 bytes character 10 bytes character
TRAN [Ⓛ]	Indicates the status of a master file record and/or the rejection of a transaction.	1 byte fixed
XTRAN [Ⓛ]	Identifies the reason for rejection of a particular transaction.	1 byte fixed

ASTATUS Flag

The ASTATUS flag indicates the status code following the execution of an array operation. It is a character field with a length of 4 bytes. The status information can be examined to determine if the operation was successful, failed, or was suppressed.

- Use the ASTATUS flag to debug applications or to determine sources of erroneous input to applications.
- As part of the array operation, there is an implied compare to blanks in the ASTATUS flag when there is an NS or GS operation immediately following an array operation. The NS or GS branch will be taken on any status other than blanks. At the “branch-to” location, the status information can be examined to determine why the operation failed or was suppressed.

The ASTATUS codes and explanations are shown in the following table.

Operations Involved					
Value	Locate Row and Column	Locate Row	Locate Column	Release	Comments
b/b/b/b	Yes	Yes	Yes	Yes	The operation executed successfully.
BMIS	Yes	Yes	No	No	The operation is unsuccessful. The value of the indicated row or column is missing or invalid as indicated or is larger than a 4-byte fixed point field. If the first character is B, the error applies to the row. If it is C, the respective error applies to the column.
BINV	Yes	Yes	No	No	
CMIS	Yes	No	Yes	No	
CINV	Yes	No	Yes	No	

Operations Involved					
Value	Locate Row and Column	Locate Row	Locate Column	Release	Comments
ROWH	Yes	Yes	No	No	<p>The operation is unsuccessful. The value supplied for row or column is not within the dimensions of the array.</p> <p>The value is greater than the highest row number.</p> <p>The row value is less than 1.</p> <p>The value is greater than the highest column number.</p> <p>The column value is less than 1.</p>
ROWL	Yes	Yes	No	No	
COLH	Yes	No	Yes	No	
COLL	Yes	No	Yes	No	
AUTO	Yes	Yes	Yes	Yes	The operation is suppressed. An automatic loop is active on the array.
INHR	Yes	Yes	Yes	Yes	The operation is suppressed. A subroutine was called; it attempted to perform an array operation on an array that was already positioned to a specific data cell due to field references in the calling request, or a subset of the array was located by a calling request by means of a successful array operation.

CHKP Flag (IMS Only)

The CHKP flag ^I triggers a checkpoint operation by placing a non-blank value into the flag from any request except preselection (type P). It is a character field with a length of 1 byte. The checkpoint does not occur until just before VISION:Builder is ready to read the next master file root segment. The flag is reset to a blank after every checkpoint.

The user can place an "A" in the CHKP flag to force an ABEND to occur instead of a checkpoint. The ABEND code issued will be 117.

CKPTID Flag (IMS Only)

The CKPTID flag ^I contains the ID of the last checkpoint taken. It is a character field with a length of 8 bytes. It contains blanks prior to the first checkpoint.

COLUMN Flag

The COLUMN flag is set after the successful completion of an array operation. It is a fixed field with a length of 4 bytes. It contains a numeric value set to the column number being processed.

- If the operation fails, the flag is set to zero.
- If an array operation is suppressed, the COLUMN flag value is not changed.

COMMAND Flag (GDBI Only)

The COMMAND flag ^G contains the command identifier. It is a character field with a length of 8 bytes. Flag values are:

Input: Sequential or Serial:	GETFIRST	Get first segment within a parent.
	GETNEXT	Get the next occurrence of the segment type previously obtained.
Key driven: (transaction driven, ⁴ ICF, start search)	GETFKEY	Get first segment with key greater than or equal to the supplied value.
	GETKEY	Get the segment with the key provided.
Output:	REPLACE ⁴	Replace the segment. Some data has been changed.
	ADD ⁴	Add the segment. A new segment has been created.
	NOCHANGE ⁴	This segment has not changed (however, BDAM may still need to see it).
	DELETE ⁴	Delete the segment. This segment was explicitly deleted by the application by a transaction or DELETE flag setting. The parent segment has not been deleted.
	INIT	Initialization request call.
	TERM	Termination request call.

The COMMAND flag settings are not strictly necessary if LM statements are used. Some mapping developers may prefer to write a single request to handle all situations. In this case, the flag becomes valuable to identify the activity required.

The information is primarily useful for directing the operation of the mapping request, but may also be useful to a database manager. The flags are initialized prior to entering a mapping request.

CONDCODE Flag

Use the CONDCODE flag to communicate with your job control. It is a binary field with a length of 2 bytes. The flag has an external effect only on MVS, CMS, and VSE.

Note: It is your responsibility to ensure that the final value of the condition code is recognized by the operating system.

You can access this flag during processing for arithmetic calculations, selection, and output. It is initialized to zero at the beginning of the run. During processing, it contains only the values supplied by you. The values normally supplied by VISION:Builder are not available during request processing.

VSE: Placing any non-zero numeric value in the CONDCODE flag cancels the job due to program control at the end of the current step. Thus, you can cancel a job by replacing 1 into CONDCODE and setting the EOF flag to all Es.

MVS and CMS: At the end of the run, after all request processing is completed and all files have reached end of file, the values you placed in the CONDCODE flag during request processing are added to the condition code values normally supplied by VISION:Builder.

- Use this procedure to control the range of condition codes without losing the settings provided by VISION:Builder.
- Only the last value of CONDCODE is used; intermediate values have no effect.
- If the CONDCODE flag is invalid at the end of the run, VISION:Builder sets it to 20 before adding it to the normal condition code value. In CMS, the condition code is referred to as the return code.

CSTATUS Flag

The CALL status flag (CSTATUS flag) indicates the status of a CALL. It is a character field with a length of 4 bytes. The CSTATUS flag values are set by VISION:Builder and indicate whether a CALL was suppressed by VISION:Builder and why.

- A CALL is suppressed if a parameter specifies an invalid or missing field. The following table shows the settings of the CSTATUS flag:

Value	Meaning
PMIS	Parameter missing.
PINV	Parameter invalid.
blank	CALL successful.

- The CSTATUS flag should be examined before using a value from the RETURNCD flag, because a suppressed call would not allow the called routine to set the RETURNCD flag.

DATE Flag

The DATE flag records the date, acquired from the operating system, in the format: MMM DD, YYYY, where MMM equals 3 alpha characters for month, DD equals 2 numerals for day of month, and YYYY equals 4 numerals for year (that is, January 15, 2001 equals JAN 15, 2001). It is a character field with a length of 12 bytes.

DELETE Flag (VISION:Builder 4000 Model Series Only)

The DELETE flag ⁴ directs VISION:Builder to delete records or segment occurrences from a master file or to force rejection of a transaction record during processing. It is a fixed field with a length of 1 byte. The settings for the DELETE flag are shown in the following table.

Setting	Result
0	All M4OLD data is output to M4NEW. The DELETE flag is automatically set to zero each time a record is processed.
1	The master file record is deleted before it can be written to the new master file following standard request processing. Records can also be dynamically deleted from the master file in type M, 2, and 3 procedures/requests. Deleted records can be output to an audit file.
2	The appropriately marked lower level segments are deleted from the new master record before it is written to the new master file. Segments are marked for deletion procedurally by placing blank or zero into the segment key and placing 2 into the DELETE flag. Segments can be marked for deletion in type N, M, 2, and 3 procedures/requests. This is not supported when processing a relational file.
4	The transaction is rejected without the master file being updated. This occurs only in type 1 and 2 procedures/requests.

ECORD Flag

The ECORD flag indicates the status of the coordinated files in relation to the master file or coordinated file to which it is chained. It is a character field with a length of 9 bytes. The values for the ECORD flag are shown in the following table.

Value	Meaning
M	The coordinated file and master file keys are equal. The coordinated file has a match and its record is available for processing.
X	The coordinated file is high in relation to the master file key. The coordinated file record is not available for processing. This value is also used when the coordinated file reaches the end of file, when no coordinated file exists for the run, or if the file is request read.
L	The coordinating file key is less than the master file key. The record is available for processing.

In the ECORD flag, each character indicates the current status of a corresponding coordinated file. Partial fielding is used to determine the status of one or more of the coordinated files, as shown below.

File Name	Partial Field Start and Length
Coordinated File 1	1,1
Coordinated File 2	2,1
Coordinated File 3	3,1
Coordinated File 4	4,1
Coordinated File 5	5,1
Coordinated File 6	6,1
Coordinated File 7	7,1
Coordinated File 8	8,1
Coordinated File 9	9,1

The ECORD flag is initialized whenever a master file record is read or created. For each coordinated file specified on an RF statement, the ECORD flag is initialized to X. As each coordinated file record is advanced, its status (in relation to the master file record being matched) is reflected in the ECORD flag.

The ECORD flag indicates file status and must not be altered. Altering the ECORD flag does not change the physical status of the files, but misleads subsequent processing.

EOF Flag

Use the EOF flag to detect or force an end of file on sequentially read input files. It is a character field with a length of 11 bytes. You can test this flag to determine if an input file exists. Alternatively, you can use it to terminate the reading of an input file by storing the appropriate value in the flag.

Note: When forcing end of file, if any value except E is placed into the EOF flag, the results are unpredictable.

Each character pertains to an input file as shown in the following table and can have one of three values:

Value	Meaning
E	File has reached end of file.
Y	File has not reached end of file.
N	File does not exist.

Input File Name	Partial Field Specification
M4OLD	1,1
M4TRAN	2,1
M4CORD1	3,1
M4CORD2	4,1
M4CORD3	5,1
M4CORD4	6,1
M4CORD5	7,1
M4CORD6	8,1
M4CORD7	9,1
M4CORD8	10,1
M4CORD9	11,1

FDNAME Flag (GDBI Only)

The FDNAME flag  contains the file name as in columns [1-8] on the FD statement. It is a character field with a length of 8 bytes.

It is initialized prior to entering a mapping request. The information in the flag is generally required by database managers.

FILE Flag (GDBI Only)

The FILE flag  contains the VISION:Builder logical file name. It is a character field with a length of 8 bytes. Values are: M4OLD, M4NEW, and M4CORD1-9.

The information is primarily useful for directing the operation of the mapping request, but can also be useful to a database manager.

FILEID Flag (GDBI Only)

The FILEID flag  contains the file identification as provided in columns [11-18] of the FD statement. It is a character field with a length of 8 bytes.

It is initialized prior to entering a mapping request. The information in the flag is generally required by database managers.

ISDATE Flag

The ISDATE flag records the system date in two International Standards Organization (ISO) formats.

Note: The date delimiter (-) is an installation option and can be changed using M4PARAMS. See the Installation Guide.

Normal processing/standard reporting (8 characters)

This flag records the date in the format of YYYYMMDD, where YYYY equals 4 numerals for year, MM equals 2 numerals for month, and DD equals 2 numerals for day (that is, January 15, 2001 equals 20010115).

Formatted reporting (10 characters)

This flag records the date in the format of YYYY-MM-DD (2001-01-15).

JULANX Flag

The JULANX flag records the system date in two formats.

Note: The date delimiter (.) is an installation option and can be changed using M4PARAMS. See the Installation Guide.

Normal processing/reporting (7 characters)

This flag records the date in the format of YYYYDDD, where YYYY equals year and DDD equals day in the year (that is, January 15, 2001 equals 2001015).

Formatted reporting (8 characters)

This flag records the date in the format of YYYY.DDD (2001.015).

JULIAN Flag

The JULIAN flag records the system date in two formats.

Note: The date delimiter (.) is an installation option and can be changed using M4PARAMS. See the Installation Guide.

Normal processing/reporting (5 characters)

This flag records the date in the format of YYDDD, where YY equals year and DDD equals day in the year (that is, January 15, 2001 equals 01015).

Formatted reporting (6 characters)

This flag records the date in the format of YY.DDD (01.015).

LILIAN Flag

The LILIAN flag contains the current date in Lilian date format as a fixed-binary number with a length of 4 bytes. The valid range of Lilian dates is 1 – 3,074,324 (October 15, 1582 to December 31, 9999). VISION:Builder automatically converts a LILIAN flag field into character format using M4PARAMS settings when a LILIAN flag field is specified for printing. The format is identical to the TODAYX flag conventions unless an override edit picture is specified.

LNUMBER FLAG

The LNUMBER flag is a text processing partial field flag that specifies the number of characters in the left part of the field. It is a fixed field with a length of 4 bytes. It is used in a scan left or right operation and can be tested, modified, or used in computation during request processing.

- The value LN can be used in place of the standard partial field specification in order to access a particular area of the scanned field.
- The value LN (dynamic partial field specification) cannot be used on the output specification (Rn) statement, nor in place of the flag names in operand A, operand B, or result specifications. For example, if LN is entered in starting character on the PR statement, the current value of LNUMBER is used at the time the PR statement is executed.

Because the value of LNUMBER may change during processing, you can use this flag to perform dynamic partial field operations.

A dynamic partial field specification might be in error during processing because:

- The partial field flag is invalid.
- The partial field specification is outside the current size of the field.
- The starting character or number of characters value is less than 1.

If the dynamic partial field specification is in error, the field to which it is applied becomes invalid for the current operation. If an invalid dynamic partial field is applied to a result field, the result field becomes invalid (except in a scan and substitute (SS) operation, where the result field is invalid for the current operation only).

LSTART Flag

The LSTART flag is a text processing partial field flag that specifies the starting position of the left part of the field. It is a fixed field with a length of 4 bytes. It is used in a scan left or right operation and can be tested, modified, or used in computation during request processing.

- The value LS can be used in place of the standard partial field specification on the processing and record selection statement in order to access a particular area of the scanned field.
- The value LS (dynamic partial field specification) cannot be used on the output specification (Rn) statement, nor in place of the flag names in operand A, operand B, or result specifications. For example, if LS is entered in starting character on the PR statement, the current value of LSTART is used at the time the PR statement is executed.

Because the value of LSTART may change during processing, you can use this flag to perform dynamic partial field operations.

A dynamic partial field specification might be in error during processing because:

- The partial field flag is invalid.
- The partial field specification is outside the current size of the field.
- The starting character or number of characters value is less than 1.

If the dynamic partial field specification is in error, the field to which it is applied becomes invalid for the current operation. If an invalid dynamic partial field is applied to a result field, the result field becomes invalid (except in a scan and substitute (SS) operation where the result field is invalid for the current operation only).

LSTATUS Flag

The LSTATUS flag indicates the status code following the execution of any segment operation. It is a character field with a length of 4 bytes. The status information can be examined to determine if the operation was successful, failed, or suppressed. This flag and its contents are independent of the type of file being accessed (for example, IMS, sequential, ISAM).

As part of the segment operation, there is an implied compare to blanks in the LSTATUS flag when there is an NS or GS operation immediately following a segment operation. The NS or GS branch will be taken on any status other than blanks. At the “branch-to” location, the status information can be examined to determine why the operation failed or was suppressed. You can use this flag in debugging applications or in determining sources of erroneous input to applications.

The LSTATUS codes and explanations are shown in the following table. The priority column indicates the order in which the LSTATUS flag settings occur.

LSTATUS Value	Applicable To			Priority	Explanation
	FS	FF FL	RS		
bbbb	Yes	Yes	Yes		The operation executed successfully.
NREC	Yes	Yes	No	1	The operation is suppressed. The current record is unavailable for the referenced segment.
AUTO	Yes	Yes	Yes	2	The operation is suppressed. The segment type referenced is already in an automatic loop; or a field in operand B is part of the segment type referenced, or part of an unreferenced (by an FS or RS operator) dependent segment.
TRAN ⁴	Yes	Yes	Yes	3	The operation is suppressed. The segment type referenced was matched, created, or inserted by the current transaction record. It can only be set in the type M procedures or its subroutines.
TDEL ⁴	Yes	Yes	Yes	3	The operation is suppressed. The segment type referenced was deleted by the current transaction record. It can only be set in the type M procedure or its subroutines.
INHR	Yes	Yes	Yes	4	The operation is suppressed. The segment type referenced in a subroutine procedure is in an automatic loop inherited from the calling procedure.
BMIS	Yes	No	No	6	A field in the operand B has a missing value.
BINV	Yes	No	No	6	A field in the operand B has an invalid value.
NPAR	Yes	Yes	Yes	7	The operation is suppressed. The segment type referenced has a parent segment that is unavailable or does not exist.
NFND	Yes	Yes	No	8	The operation fails. VISION:Builder cannot find any segment occurrences that satisfy the operation.
NOMO	Yes	No	No	8	The operation fails. The next occurrence of a referenced segment type does not exist.

M4AUDIT Flag (VISION:Builder 4000 Model Series Only)

The M4AUDIT flag ⁴ provides access to the number of records output to the M4AUDIT file during application execution. It is a fixed field with a length of 4 bytes. The M4AUDIT flag can be referenced in any type of request (except preselection); however, it can only be referenced if the M4AUDIT file is being used in the application.

M4CORDn (n=1 to 9) Flags

There are separate flags provided for each one of the nine VISION:Builder coordinated files, M4CORDn where n=1 to 9. These are fixed fields with lengths of 4 bytes.

These flags (M4CORD1, M4CORD2, M4CORD3, M4CORD4, M4CORD5, M4CORD6, M4CORD7, M4CORD8, and M4CORD9) provide access to the number of records read from each of the various coordinated files during application execution. These coordinated file flags can be referenced in any type of request (except preselection); however, they can only be referenced if the coordinated file they reference is being used in the application.

M4NEW Flag (VISION:Builder 4000 Model Series Only)

The M4NEW flag ⁴ provides access to the number of records output to the M4NEW file during application execution. It is a fixed field with a length of 4 bytes. The M4NEW flag can be referenced in any type of request (except preselection); however, it can only be referenced if the M4NEW file is being used in the application.

M4OLD Flag

The M4OLD flag provides access to the number of records read from the M4OLD file during application execution. It is a fixed field with a length of 4 bytes. The M4OLD flag can be referenced in any type of request (except preselection); however, the flag can only be referenced if the M4OLD file is being used in the application. When using update-in-place, the flag will represent only the records read from M4OLD.

M4REJECT Flag (VISION:Builder 4000 Model Series Only)

The M4REJECT flag ⁴ provides access to the number of records output to the M4REJECT file during application execution. It is a fixed field with a length of 4 bytes. The M4REJECT flag can be referenced in any type of request (except preselection); however, it can only be referenced if the M4REJECT file is being used in the application.

M4SUBFn (n= 0 to 9) Flags

There are separate flags provided for each one of the ten VISION:Builder subfiles, M4SUBFn where n=0 to 9. These are fixed fields with lengths of 4 bytes.

These flags (M4SUBF0, M4SUBF1, M4SUBF2, M4SUBF3, M4SUBF4, M4SUBF5, M4SUBF6, M4SUBF7, M4SUBF8, and M4SUBF9) provide access to the number of records output to each of the various subfiles during application execution. These subfile flags can be referenced in any type of request (except preselection); however, they can only be referenced if the subfile they reference is being used in the application.

M4TRAN Flag (VISION:Builder 4000 Model Series Only)

The M4TRAN flag ⁴ provides access to the number of records read from the M4TRAN file during application execution. It is a fixed field with a length of 4 bytes. The M4TRAN flag can be referenced in any type of request (except preselection); however, it can only be referenced if the M4TRAN file is being used in the application.

MISSPASS Flag

The MISSPASS flag value is automatically set by VISION:Builder only at the start of the all-miss pass (ECORD setting of all Xs). It is a character field with a length of 1 byte. This flag will contain either an X or an M during the all-miss pass. An X setting indicates that no hits occurred during coordination for the master record; an M setting indicates at least one hit occurred during coordination. The MISSPASS flag is blank when the all-miss pass is not in progress.

While the ECORD flag indicates the status of coordinated files, the MISSPASS flag indicates the status of the master file. The MISSPASS flag should not be used with RPCORDONLY, because its value during a match or low cycle is blank and meaningless.

MNUMBER Flag

The MNUMBER flag is a text processing partial field flag that specifies the number of characters in the middle (matching) part of the field. It is a fixed field with a length of 4 bytes. It is used in a scan left or right operation and can be tested, modified, or used in computation during request processing.

- The value MN can be used in place of the standard partial field specification in order to access a particular area of the scanned field.
- The value MN (dynamic partial field specification) cannot be used on the output specification (Rn) statement, nor in place of the flag names in operand A, operand B, or result specifications. For example, if MN is entered in starting character on the PR statement, the current value of MNUMBER is used at the time the PR statement is executed.

Because the value of MNUMBER may change during processing, you can use this flag to perform dynamic partial field operations.

A dynamic partial field specification might be in error during processing because:

- The partial field flag is invalid.
- The partial field specification is outside the current size of the field.
- The starting character or number of characters value is less than 1.

If the dynamic partial field specification is in error, the field to which it is applied becomes invalid for the current operation. If an invalid dynamic partial field is applied to a result field, the result field becomes invalid (except in a scan and substitute (SS) operation where the result field is invalid for the current operation only).

MODE Flag (GDBI Only)

The MODE flag[Ⓞ] contains information about the application mode of operation. The following table shows the X and Y values. It is a character field with a length of 2 bytes.

In Operation Mode X =		In Update Mode Y =	
S	Standard Processing	R	Retrieval Update
M	MOSAIC Processing	U	Update
		L	Load Module

It is initialized prior to entering a mapping request. The information in the flag is generally required by database managers.

MSTART Flag

The MSTART flag is a text processing partial field flag that specifies the starting position of the middle (matching) part of the field. It is a fixed field with a length of 4 bytes. It is used in a scan left or right operation and can be tested, modified, or used in computation during request processing.

- The value MS can be used in place of the standard partial field specification on the processing and record selection statement in order to access a particular area of the scanned field.
- The value MS (dynamic partial field specification) cannot be used on the output specification (Rn) statement, nor in place of the flag names in operand A, operand B, or result specifications. For example, if MS is entered in starting character on the PR statement, the current value of MSTART is used at the time the PR statement is executed.

Because the value of MSTART may change during processing, you can use this flag to perform dynamic partial field operations.

A dynamic partial field specification might be in error during processing because:

- The partial field flag is invalid.
- The partial field specification is outside the current size of the field.
- The starting character or number of characters value is less than 1.

If the dynamic partial field specification is in error, the field to which it is applied becomes invalid for the current operation. If an invalid dynamic partial field is applied to a result field, the result field becomes invalid (except in a scan and substitute (SS) operation where the result field is invalid for the current operation only).

MSTATUS Flag (GDBI Only)

The MSTATUS flag  is used by the mapping request to instruct VISION:Builder on a specific action to take subsequent to completion of the mapping request. It is a character field with a length of 6 bytes.

MSTATUS is initialized to blanks prior to each call to a mapping request. Permissible return values are listed in the following table.

Value	Explanation
blank	Mapping successfully completed.
NFOUND	Segment not found. Useful for input mapping only. Signifies to VISION:Builder that there was no data to fill the skeleton segment, either because of having run out of repeated segments, or because of failure to locate a keyed segment. In the process of building a hierarchical record (standard or MOSAIC processing, non-keyed operations), VISION:Builder returns to the mapping requests for each segment type to request repeated segments. Thus, the mapping requests must be programmed to issue NFOUND for each segment type to terminate a series of repeated segments.
STOPnn	Stop the run and set the condition code to nn (condition code setting support in MVS only). Used by mapping requests to cease processing.

The information is primarily useful for directing the operation of the mapping request, but can also be useful to a database manager.

OWN Flag

The OWN flag is used by own-code routines to communicate between own-code requests. It is a character field with a length of 16 bytes.

It is not examined or altered by VISION:Builder in any way. The flag name OWN can be entered in field name A, B, or C depending on the operation being performed.

PAGE Flag

The PAGE flag is used only in formatted reporting and specifies the placement of the page numbers, which are printed left-aligned in the title and summary lines of formatted reports. It is a character field with a length of 6 bytes. Enter the PAGE flag as F.PAGE.

PASSWORD Flag (GDBI Only)

The PASSWORD flag ^G contains the password as provided on the RF statement for the appropriate VISION:Builder file. It is a character field with a length of 8 bytes. It is available for all mapping requests, but is expected to be most valuable to the initialization mapping request to control access to the database. This field is set prior to a mapping request receiving control. Valid only in mapping requests.

RESTART Flag (IMS Only)

The RESTART flag ^I is used by requests to determine the restart status of a VISION:Builder run. It is a character field with a length of 8 bytes. If a run has been restarted, the RESTART flag contains the checkpoint ID at which the restart occurred. Otherwise, the value of the RESTART flag is all blanks.

RETURNCD Flag

The return code (RETURNCD flag) reflects the status of a CALL. It is a fixed field with a length of 4 bytes. The values are set by the called routine and the contents can be examined after a CALL. If the flag is included in a report, VISION:Builder treats the flag as a 2-byte fixed point field (output width of 7, maximum printable value 99,999). The value in this field is the value in general register 15 upon return to VISION:Builder from a called routine.

RNUMBER Flag

The RNUMBER flag is a text processing partial field flag that specifies the number of characters in the right part of the field. It is a fixed field with a length of 4 bytes. It is used in a scan left or right operation and can be tested, modified, or used in computation during request processing.

- The value RN can be used in place of the standard partial field specification in order to access a particular area of the scanned field.
- The value RN (dynamic partial field specification) cannot be used on the output specification (Rn) statement, nor in place of the flag names in operand A, operand B, or result specifications. For example, if RN is entered in starting character on the PR statement, the current value of RNUMBER is used at the time the PR statement is executed.

Because the value of RNUMBER may change during processing, you can use this flag to perform dynamic partial field operations.

A dynamic partial field specification might be in error during processing because:

- The partial field flag is invalid.
- The partial field specification is outside the current size of the field.
- The starting character or number of characters value is less than 1.

If the dynamic partial field specification is in error, the field to which it is applied becomes invalid for the current operation. If an invalid dynamic partial field is applied to a result field, the result field becomes invalid (except in a scan and substitute (SS) operation where the result field is invalid for the current operation only).

ROW Flag

The ROW flag is set after successfully completing an array operation. It is a fixed field with a length of 4 bytes. It contains a numeric value set to the ROW number being processed. If the operation fails, the flag is set to zero. If an array operation is suppressed, the ROW flag is not suppressed.

RSTART Flag

The RSTART flag is a text processing partial field flag that specifies the starting position of the right part of the field. It is a fixed field with a length of 4 bytes. It is used in a scan left or right operation and can be tested, modified, or used in computation during request processing.

- The value RS can be used in place of the standard partial field specification on the processing and record selection statement in order to access a particular area of the scanned field.
- The value RS (dynamic partial field specification) cannot be used on the output specification (Rn) statement, nor in place of the flag names in operand A, operand B, or result specifications. For example, if RS is entered in starting character on the PR statement, the current value of RSTART is used at the time the PR statement is executed.

Because the value of RSTART may change during processing, you can use this flag to perform dynamic partial field operations.

A dynamic partial field specification might be in error during processing because:

- The partial field flag is invalid.
- The partial field specification is outside the current size of the field.
- The starting character or number of characters value is less than 1.

If the dynamic partial field specification is in error, the field to which it is applied becomes invalid for the current operation. If an invalid dynamic partial field is applied to a result field, the result field becomes invalid (except in a scan and substitute (SS) operation where the result field is invalid for the current operation only).

RSTATUS Flag

The RSTATUS flag indicates the results of the most recent request-read (RD, RE, or RG) operation performed in a PR statement. It is a character field with a length of 4 bytes.

The RSTATUS flag is set whenever a read (RD), read equal key (RE), or read greater than or equal key (RG) operator is used with coordinated files. The settings of the RSTATUS flag are shown in the following table.

Value	Explanation
AINV	Operand A contains an invalid field. The RE or RG operation was not performed.
AMIS	Operand A was missing. The RE or RG operation was not performed.
KEQL	The RE or RG operation was performed and a root segment with a key equal to the key in operand A was successfully returned, or an RD was successful.
KGTR	The RG operation was performed and a root segment with a key greater than the key in operand A was successfully returned.
NREC	The operation was performed but no record was found.
RIGN	The read operation was ignored. A segment in the specified coordinated file is in a loop or under control of a find segment (FS) operation.
EGRP	Specifies the end of a group. A sequential RD operator following a direct-read operator compares the key of each root segment retrieved to the search argument for the direct-read. When the key no longer matches, the RSTATUS flag indicates the end of group. EGRP is also set if end of file is reached on a sequential RD operation. In this situation, the EOF flag for the appropriate coordinated file is set to E and reset to Y on the next RE or RG operation for that file.

This is the only situation in which VISION:Builder resets an EOF flag value. Successive RD operations without any intervening RE or RG operations continue to return the EGRP and EOF flag values.

An NS or GS statement immediately following an RE or RG operator causes an implied compare of the RSTATUS flag.

- If the RE or RG operation fails (that is, the RSTATUS flag is equal to AINV, AMIS, or NREC), the NS branch is taken.
- If the RE or RG operation is successful (that is, the RSTATUS flag is equal to KEQL, KGTR, or RIGN), processing continues with the next instruction.

You can use the GS statement in the same way as the NS statement to determine success or failure.

SEGNAME Flag (GDBI Only)

The SEGNAME field [Ⓢ] contains the segment name as defined in columns [11-18] of the LS statement. It is a character field with a length of 8 bytes. If an alias segment name is provided on an LB statement in a run data group, the SEGNAME flag still refers to the segment name as defined on the LS statement.

The information is primarily useful for directing the operation of the mapping request, but can also be useful to a database manager. It is initialized prior to entering a mapping request.

SQL Flag (DB2 Only)

The SQL flag [Ⓢ] contains status information after an attempt is made to insert a row into a DB2 table with a UNIQUE index. It is a character field with a length of 4 bytes. It contains the character string DUPL if a matching row already exists in the table. It contains blanks if a matching row does not exist.

- If F.SQL equals blanks, the record is successfully inserted.
- If F.SQL equals DUPL and Scan/Terminate does not equal 4, the run terminates; however, if Scan/Terminate equals 4, processing continues (the rejected row can be output to a report or another subfile). Note that this is the only instance in which the Scan/Terminate value determines processing based on message types other than type 4.
- If type 0 and type 1 messages are suppressed, there is no indication that the attempt to insert the row failed.

SSCOUNT Flag

The SSCOUNT flag counts the number of matches found during the REPLACE operation. It is a fixed field with a length of 2 bytes. The flag is set to zero at the start of each run. The settings are shown in the following table.

Setting	Result
0	There are no successful substitutes or the search-string is longer than the field to be modified.
-1	Used with variable length fields. If a REPLACE operation causes the maximum length of a type V field to be exceeded, the operation does not take place, and the field to be modified is made invalid.
-2	The field to be modified or search-string is null and/or invalid, and/or the replace-string is invalid prior to a REPLACE operation.

STRAN Flag (VISION:Builder 4000 Model Series Only)

The STRAN flag  interrogates the status of the current transaction at each segment level and indicates the actions applied to the master file record by the current transaction. It is a character field with a length of 9 bytes. The STRAN flag is used in type M requests and their subroutines, but is available to any request type.

Each of the 9 bytes of the STRAN flag corresponds to a segment level in a structured record. The settings are shown in the following table.

Setting	Explanation
Blank	No action at this level.
M	Match actions only occurred at this level. All match fields for the segment at this level were matched (transaction action code M), but no fields in the segment were updated (action codes B, R, A, S, P, or X).
I	A segment was inserted at this level. This setting applies to level 1 (root), as well as lower levels 2 through 9. Thus, action code C or I was used or M with default create or insert specified. In addition, update actions (codes B, R, A, S, P, or X) may have been applied after the create or insert.
U	Update actions (codes B, R, A, S, P, or X) were applied to the segment at this level, but C or I was not specified.
D	The segment at this level was deleted by a transaction (action code E). This setting applies only to the level that was explicitly deleted by the transaction, not to lower level dependent segments. This setting is not apparent at level 1, because deleted master records are not presented to any type of request.

The STRAN flag is reset to blanks before obtaining the next logical transaction record (that is, either reading a new transaction record or retrying the last one). However, the STRAN flag is not revalidated if you set it to an invalid value. This is consistent with the treatment of other user flags.

The STRAN flag is set to non-blank values as the transaction actions are successfully applied.

If a transaction is referenced in the type 4 request and rejected because of update errors, the STRAN flag reflects the match actions that occurred but does not reflect the insert, update, or delete actions. These actions were either not yet applied or were backed out before the type 4 request received control.

TIME Flag

The TIME flag records the time of day at which the run was started. It is a character field with a length of 8 bytes. The TIME flag prints as HH.MM.SS, where HH equals hours, MM equals minutes, and SS equals seconds. Leading zeroes are added where necessary.

TODAY Flag

The TODAY flag records the system date in two formats.

Note: The order of month, day, and year and the date delimiter (/) are installation options and can be changed using M4PARAMS. See the Installation Guide.

Normal processing/standard reporting (6 characters)

The TODAY flag records the date in the format MMDDYY, where MM equals 2 numerals for month, DD equals 2 numerals for day, and YY equals 2 numerals for year (that is, January 15, 2001 equals 011501).

Formatted reporting (8 characters)

The TODAY flag records the date in the format MM/DD/YY (01/15/01).

TODAYX Flag

The TODAYX flag records the system date in two formats.

Note: The order of month, day, and year and the date delimiter (/) are installation options and can be changed using M4PARAMS. See the Installation Guide.

Normal processing/standard reporting (8 characters)

The TODAYX flag records the date in the format MMDDYYYY, where MM equals 2 numerals for month, DD equals 2 numerals for day, and YYYY equals 4 numerals for year (that is, January 15, 2001 equals 01152001).

Formatted reporting (10 characters)

The TODAYX flag records the date in the format MM/DD/YYYY (01/15/2001).

TRAN Flag (VISION:Builder 4000 Model Series Only)

The TRAN flag ^④ indicates the maintenance status of a master file record and/or the rejection of a transaction record. It is a fixed field with a length of 1 byte. The settings are shown in the following table.

Value	Conditions
0	No transaction records exist for the master file record.
1	The master file record was updated or matched by a transaction record.
2	The master file record was created by a transaction record.
3	Conditions for values 1 and 2 both apply.
4	One or more transaction records against this master file record were rejected.
5	Conditions for values 1 and 4 both apply.
6	Conditions for values 2 and 4 both apply.
7	Conditions for values 1, 2, and 4 all apply.
8 or higher	Invalid.

XTRAN Flag (VISION:Builder 4000 Model Series Only)

The XTRAN flag ^④ identifies the reasons for rejected transactions. It is a fixed field with a length of 1 byte. When more than one error is detected in a transaction, only the last one is described.

The XTRAN settings are shown in the following table.

Value	Explanation
0	Inactive setting.
1	Request rejection.
2	The transaction record does not match any of the defined identifiers.
3	The field in the transaction is outside the maximum or minimum values specified in the transaction definition.
4	A transaction record key is less than the preceding transaction record key; the transaction file is out of sequence.
5	Transaction record key field with M, D, or C action cannot be converted for matching purposes.

Value	Explanation
6	The contents of the date field in the transaction do not meet the date validation criteria.
7	The field in the transaction record does not conform to the input edit pattern specified in the transaction definition.
8	The transaction record key does not match any master file key and a create is not specified.
9	A create action has been specified for a record that already exists on the master file.
10	Field associated with an M, E, C, D, or I action cannot be converted for matching purposes or a field associated with B, R, A, S, P, or X action causes an arithmetic or conversion error.
11	A non-record key field (that is, any field other than a record key field) associated with an M action cannot be located in the master file record.
12	The segment for which a delete segment action (E) is specified does not exist in the master file record.
13	The insert action (I) has been specified for a segment that is repeated a fixed number of times. There are no "empty" segments into which the insert can be made.
14	A segment to be inserted already exists in the master file record.
15	This value occurs with variable length records only. The requested insertion would cause the maximum record length to be exceeded.
17	The count field controlling the segment to be inserted is too small to contain the number of subordinate segments being inserted.

Conversion Functions

This appendix provides suggestions for converting existing VISION:Builder requests to ASL procedures.

Note: You must have experience with the fixed format PR statements.

Examples are provided in [Conversion Examples on page E-4](#). The examples will clarify certain concepts, demonstrate particular capabilities, or offer practical techniques. Because some of these examples were coded and run in previous releases, forms and listings may not conform to current release formats. These minor discrepancies have no effect on the validity or value of the examples in relation to ASL.

The intention of the manual is not to describe the specific use and operation of ASL and assumes that the reader is already familiar with VISION:Builder Fixed Syntax Processing and Record Selection (PR) statements.

Why ASL?

ASL is a free-form specification language with a comprehensive vocabulary and syntax that gives you capabilities to express decision-making logic beyond the fixed format VISION:Builder syntax. ASL streamlines many of the elements of VISION:Builder code.

Defining ASL Procedures

ASL is generally used in VISION:Workbench for DOS and the VISION:Workbench for ISPF. However, ASL procedures can also be entered in-stream with fixed format VISION:Builder requests. Procedure statements and fixed format statements cannot be mixed in the same procedure or request. However, an application may contain a mixture of fixed format syntax and ASL procedures (for example, separate requests).

In-stream ASL statements following any fixed-syntax statements are entered after a `;;BEGASL` statement. An `;;ENDASL` statement must be entered after the last ASL statement that is in turn followed by fixed-syntax statements.

The following is an example of in-stream ASL statements:

```
run control group (fixed-syntax)
.
.
;;BEGASL
PROC
.
.
ASL statements
.
.
END PROC
;;ENDASL
reaname ER
PR statements
.
.
;;BEGASL
PROC
.
.
ASL statements
.
.
END PROC
```

Tips and Techniques

ASL procedures and fixed format VISION:Builder requests can be used in the same application. As you perform program maintenance, use this opportunity to begin replacing fixed format statements with ASL procedures. You cannot use PR statements and ASL commands in the same request but you can call an ASL procedure from a fixed format VISION:Builder request.

When converting an application, do not attempt to translate each statement on a one for one basis. The function of the code is more important than matching exactly how it is done. Remember, ASL lends itself to a building block structure that is readable in a simple top-down fashion.

VISION:Workbench for DOS is an invaluable tool for learning ASL. It provides syntax checking during the data entry process and includes a validation function. One way to become comfortable with ASL is to actually go through a complete VISION:Workbench export and specify that the ASL be translated into fixed format syntax. By reviewing the fixed format VISION:Builder code created by this type of export process, you can verify that what you wrote in ASL translates into what you want to accomplish. Once you are comfortable with ASL, your application can be exported in ASL.

ASL Syntax and Terminology

ASL is a free-form language consisting of labels, commands, and functions. A procedure statement begins on a new line and consists of an optional label followed by a command. If the statement has a label, the label must be followed immediately by a colon. What appears on the rest of the procedure statement after the command depends on the syntax of the command. For example:

```
LABELX: LET FIELDA = FIELDB
```

is an actual procedure statement where:

LABELX	is the statement label.
LET	is the command.
FIELDA = FIELDB	places the contents of field B into field A (FIELDA and FIELDB are the operands).

One or more spaces must separate the command, keywords, and operands. For example:

```
CUSTNAME: FIELD TYPE C LENGTH 30 HEADING 'CUSTOMER'
```

These examples are equally valid FIELD command procedure statements that define a temporary field named CUSTNAME (the statement label is the field name).

Procedure statements can be written using multiple lines. Each continuation statement must be terminated by a comma *preceded* by a blank and then continued on the next line. For example, both of the following statements are syntactically correct:

```
IF NAME = 'ABC COMPANY' ,
  AND NUMBER = '01' ,
  OR NUMBER = '02'
```

or

```
IF NAME = 'ABC COMPANY' AND NUMBER = '01' OR NUMBER = '02'
```

[Conversion Table on page E-14](#) contains a guide to the more common operations used in fixed format VISION:Builder processing (PR) statements and relates them to their approximate ASL function or command. The full syntax of ASL is explained in the VISION:Builder ASL Reference Manual.

Indentation is provided only for program readability and does not affect execution of an ASL statement. To improve readability, start the words IF/DO CASE and ELSE in the same line position and indent the action statement verbs. Place the word ELSE on a line by itself. The END command should be on a line by itself and aligned with the IF or DO command that initiates the block of code delimited by the END. An END command is required for every DO or IF command. It is easier to determine that all delimiters have been accounted for with indentation. In

addition, program modifications and debugging often require modification to action statements. By indenting DO and IF blocks, such changes can be made with minimum code disruption and chance for error.

In ASL, comments can be placed anywhere following a semicolon (;). Comments are used throughout the examples shown in [Conversion Examples on page E-4](#).

For easy identification of fields, a standard one-character qualifier and a period can prefix a name. A qualifier identifies a specific file or usage of a field. For example, T.TEMP is a reference to a temporary field named TEMP and 1.CORDFLD is a reference to a field named CORDFLD from the file in the application that has been assigned to coordinated file 1. If a qualifier does not precede a field name, the field is located in the new master file.

If a name contains an embedded blank or special character, the name must be enclosed in double quotation marks. For example:

```
T."CORD-FLD"  
1."LAST AMT"
```

Conversion Examples

This chapter contains simple and complex conversion examples. The following are some simple examples.

Objective: Add several fields together with rounding.

ASL:

```
LET RESULT = FIELDA + FIELDB + FIELDC + FIELDDD + FIELDE ,  
            WITH ROUNDING
```

Fixed Format:

```
FIELDA + FIELDB           RESULT  
RESULT + FIELDC          RESULT  
RESULT + FIELDDD         RESULT  
RESULT + FIELDE          RESULT  
RESULT + D.005           RESULT
```

Objective: Combine a field date into one with month, day, and year with editing (for example, 101299 converted to 10/12/99).

ASL:

```
COMBINE PF (DATE 1 2) '/' PF (DATE 3 2) '/' PF (DATE 5 2) STORE RESULT
```

Fixed Format:

```
DATE      R                      RESULT  0102A
RESULT    COC/                   RESULT
RESULT    CO DATE                RESULT  0302A
RESULT    COC/                   RESULT
RESULT    CO DATE                RESULT  0502A
```

Objective: Report employees who earn more than their respective managers.

ASL:

```
IF SALARY GT MGRSAL THEN
  CALL REPORT HIGHPAID
END
```

Fixed Format:

```
SALARY  GT MGRSAL
        NS END
        GO SUB HIGHPAID
```

Example 1 Interest on Overdue Payments

The accounts receivable department needs to prepare bills for overdue payments. Interest is charged on all overdue payments at a rate of 1.5% per month. The incoming file contains the payment due date and the payment principal amount.

The following are the steps to calculate the interest.

- Calculate the number of months the payment is overdue.
- Set payment due amount to the principal payment amount.
- Start a loop to set payment due to the principal amount multiplied by 1.015.
- Decrement the overdue month counter by 1. Continue looping until this counter is zero (0).

Figure E-1 shows the ASL procedure for Example 1. Note that all dates in this example are in the format MMDDYY.

```

*****
*   PROC NAME - CALCINT   *
*   INPUT STREAM PROCEDURE *
*****
CALCINT: PROC TYPE SUBROUTINE
;CALCULATE INTEREST FOR PAYMENT
;
;Define temporary fields
;
YY:      FIELD F 4
MM:      FIELD F 4
NEWAMT:  FIELD P 8 2
INTEREST: FIELD P 8 2
;
;Compute the number of months over which interest is to be charged
;
LET T.YY = PF(F.TODAY 5 2) - PF(DUEDATE 5 2) ;Compute years overdue
IF T.YY GE 0 ;Due date this year or later?
  LET T.MM = T.YY * 12 + ,
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
```

Figure E-1 Example 1: ASL Procedure

```

          PF(F.TODAY 1 2) - PF(DUEDATE 1 2) ;Compute months overdue
LET T.NEWAMT = PRINCIPL
DO WHILE T.MM GT 0 ;Accumulate new amount
  LET T.NEWAMT = T.NEWAMT * 1.015
  LET T.MM = T.MM - 1
END
LET T.INTEREST = T.NEWAMT - PRINCIPL ;Compute actual interest
END
END PROC

```

Figure E-1 Example 1: ASL Procedure

The first LET statement calculates the elapsed years between the current year (F.TODAY) and the master file field DUEDATE year. The IF command in this example has no associated ELSE. If the temporary field (T.YY) is greater than or equal to zero the IF group is processed. If the temporary field YY is less than zero, the logical expression is false and control is transferred to the corresponding END statement for this IF block. The normal record processing cycle continues with the next occurrence of the segment controlling the record domain.

IF T.YY is greater than or equal to zero, the logical expression is true. The number of elapsed months is calculated by multiplying the number of years by 12 and adding the difference between the current month and the due date month. The current principal amount is copied to the temporary field NEWAMT.

The DO WHILE logical expression establishes an explicit loop that is evaluated at the beginning of each iteration if T.MM is greater than zero before each pass through the calculation.

When the loop terminates, T.INTEREST is determined by subtracting the original principal amount from T.NEWAMT.

[Figure E-2](#) shows the fixed format VISION:Builder statements for Example 1.

Example 1 illustrates the steps of the sample problem in ASL with the result that the functionality of the fixed format VISION:Builder code remains, but the ASL code has a more closed-end decision construct. It has a smooth sequential flow that is easier to understand than fixed format syntax.

```

*****
* REQUEST NAME - CALCINT *
* INPUT STREAM REQUEST *
*****

```

STMT REPORT	REQUESTOR ID	MAX SEL SUM VERT	FORMS PAGE PAGE	LINE REQ	BACK BR RNT AB	SEQUENCE	LIST	
TYPE DATE		ITEMS CTL RPT SP	CNTRL WDIH HGHT NOS?	TYP SET NAME	CONTROL TMP FM	73 TO 80	SEQ	
(ER)				(S)			9	
STMT	FIELD	FIELD FLD DEC	OUTPT EDIT	INITIAL	LINE 1 OF	LINE 2 OF	SEQUENCE	LIST
TYPE	NAME	LENGTH TYP PLC EDIT	LGTH	VALUE	COLUMN HEADING	COLUMN HEADING	73 TO 80	SEQ
(TF)	(YY)	(4) (E)						10
(TF)	(MM)	(4) (E)						11
(TF)	(LOOP)	(4) (E)						12
(TF)	(NEWAMT)	(8) (P) (2)						13
(TF)	(INTEREST)	(8) (P) (2)						14
STMT SEQ	LOG CONOPERAND-A.....	OPEROPERAND-B.....RESULT.....	PARTIAL FIELD	SEQUENCE	LIST
TYPE NO.	LEV CTR QLF	FIELD SEG-LVL	AIN	QLF FIELD SEG-LVL (OR CONSTANT)	QLF FIELD SEG-LVL	STRT NUMB OPR	73 TO 80	SEQ
(PR) (010)		(F,TODAY)	(R)		(T,YY)	(5) (2) (A)		15
(PR) (020)		(T,YY)	(-)	(DUEDATE) 01-1	(T,YY)	(5) (2) (B)		16
(PR) (030)		(T,YY)	(GE)	(D,0)				17
(PR) (040)			(NS)	(END)				18

Figure E-2 Example 1: Fixed Format Request

heading will appear over the column of data when the field is output to a report. All of the fixed format VISION:Builder temporary field (TF) statement entries are available in ASL and all of the data entry rules apply as well.

The ASL procedure has two IF statements. The special relational operator > has been used in place of the operator GT; either relational operator is permitted.

- The first IF block checks the master file field HOURS to see if it is greater than 40. If it is, the temporary field OTHOURS is set to the number of hours in excess of 40.
- The second IF block of statements checks the master file field OTHOURS to see if it is greater than zero. If it is, the value in the master file field OTHOURS is added to the value in the temporary field OTHOURS and the record is selected for output.

[Figure E-4](#) shows the request OTIMCARD.

```

*****
* REQUEST NAME - OTIMCARD *
* INPUT STREAM REQUEST *
*****

-----
SMT REPORT          MAX SEL SUM VERT FORMS PAGE PAGE LINE REQ  BACK BR RNT AB  SEQUENCE LIST
TYPE DATE          REQUESTOR ID  ITEMS CTL RPT SP  CNTRL WIDTH HGT NOS? TYP SET NAME CONTROL TMP EM  73 TO 80  SEQ
-----
(E)                                                         (S)                                                         9
-----
SMT  FIELD  FIELD FLD DEC OUTPT EDIT  INITIAL          LINE 1 OF          LINE 2 OF          SEQUENCE LIST
TYPE  NAME  LENGTH TYP PLC EDIT  LGTH             VALUE             COLUMN HEADING    COLUMN HEADING    73 TO 80  SEQ
-----
(TF) (OTHOUS ) ( 3) (Z) (L)                                                         10
-----
SMT SEQ LOG CON .....OPERAND-A..... OPER .....OPERAND-B..... .....RESULT..... PARTIAL FIELD SEQUENCE LIST
TYPE NO. LEV CTR QLF FIELD  SEG-LVL  AIN  QLF FIELD  SEG-LVL  (OR CONSTANT)  QLF FIELD  SEG-LVL  STRT NUMB OPR  73 TO 80  SEQ
-----
(E) (010)          (HOURS ) 01-1  (GT)  (D,40          )                                     )                                     11
(E) (020)          (HOURS ) 01-1  (NS)  (040          )                                     )                                     12
(E) (030)          (HOURS ) 01-1  (- )  (D,40          ) (T,OTHOUS )                                     )                                     13
(E) (040)          (OTHOUS ) 01-1  (GT)  (D,0          )                                     )                                     14
(E) (050)          (HOURS ) 01-1  (NS)  (070          )                                     )                                     15
(E) (060)          (OTHOUS ) 01-1  (+ )  (T,OTHOUS ) (T,OTHOUS )                                     )                                     16
(E) (070)          (HOURS ) 01-1  (GO)  (SUB OTHOURS ) (T,OTHOUS )                                     )                                     17

```

Figure E-4 Example 2: Fixed Format Request

Example 3 Country by Sports Category

Note: Example 3 converts an array application. Even if your installation does not have or does not use the array feature, this example provides some significant illustrations of ASL syntax.

The Olympic committee wants to track the performance of selected countries in selected sports categories. To accomplish this task, you need a cross-tab report showing the number of gold, silver, and bronze medals won by each of the 20 selected countries in each of the six selected sports categories. Also, calculate and report the percentage of medals for each country and final totals and percentages.

[Figure E-6](#) through [Figure E-8](#) show the program written in ASL. [Figure E-9 on page E-11](#) shows the program written using PR statements.

The program is to be written using a structured, modular approach.

- Define a two dimensional array with six sports and 20 countries. For the medal counts, each cell has three fields: gold, silver, and bronze. [Figure E-5](#) shows an example of the array's layout.

	UNITED STATES			GERMANY			CANADA		
SKATING	G	S	B	G	S	B	G	S	B
SKIING	G	S	B	G	S	B	G	S	B
LUGE	G	S	B	G	S	B	G	S	B

Figure E-5 Example 3: Array Layout

- Read the input file. Locate the correct row for the sport and column for the country. Add one to the appropriate medal count. (See the procedure BUILD in [Figure E-6](#).)
- After all records have been read, define the temporary fields that will contain the counts and percentages. (See the procedure PRNTCNTL in [Figure E-7](#).)
- Read each row of the array and set the temporary counters to zero. (See the DO WHILE loop in the procedure PRNTCNTL in [Figure E-7](#).)
- Call a procedure to add the totals to the temporary fields. (See the procedure TOTAL in [Figure E-7](#).)
- Call a procedure to calculate the percentages and print the report. (See the procedure PRINT in [Figure E-8](#).)

```

*****
* PROC NAME - BUILD *
* INPUT STREAM PROCEDURE *
*****

BUILD: PROC
;ADD RESULT INTO ARRAY
;
;This procedure sums the results into the array by:
;(1) type of medal, (2) type of sport, (3) country.
;
IF CNTYNO LT 21 AND SPORNO LT 7
LOC A ROW SPORNO COLUMN CNTYNO
DO CASE
CASE WHERE MEDAL EQ 'GOLD'
LET A.GOLD = A.GOLD + 1
CASE WHERE MEDAL EQ 'SILV'
LET A.SILVER = A.SILVER + 1
CASE WHERE MEDAL EQ 'BRON'
LET A.BRONZE = A.BRONZE + 1
END
END
END PROC
    
```

Figure E-6 Example 3: ASL Procedure BUILD

The ASL procedure BUILD provides an excellent example of the use of the CASE command, as well as the simplified top-down structure of ASL. Only the first group of procedure statements whose CASE command condition is true is performed within the DO CASE block. Once the proper counter has been incremented, control is transferred immediately to the END command.

```

*****
* PROC NAME - PRNTCNTL *
* INPUT STREAM PROCEDURE *
*****

PRNTCNTL: PROC TYPE EOF, TEMPREINIT
;ROW LOOPING DRIVER
;
;This procedure sets up a loop of 6 to cause all rows
;of the array to be processed one row at a time.
;
COUNTRY: FIELD C 15 HEAD 'COUNTRY'
SPORT: FIELD C 15 HEAD 'SPORT'
TOTGOLD: FIELD F 4 INIT 0 HEAD 'TOTAL' 'GOLD'
TOTSILV: FIELD F 4 INIT 0 HEAD 'TOTAL' 'SILVER'
TOTBRON: FIELD F 4 INIT 0 HEAD 'TOTAL' 'BRONZE'
TOTMED: FIELD F 4 INIT 0 HEAD 'TOTAL' 'MEDALS'
GRANDTOT: FIELD F 4 INIT 0 HEAD 'GRAND' 'TOTAL'
PCTGOLD: FIELD F 4 2 INIT 0 HEAD 'PCT OF' 'GOLD'
PCTSILV: FIELD F 4 2 INIT 0 HEAD 'PCT OF' 'SILVER'
PCTBRON: FIELD F 4 2 INIT 0 HEAD 'PCT OF' 'BRONZE'
PCTTOT: FIELD F 4 2 INIT 0 HEAD 'PCT OF' 'TOTAL'
ROW: FIELD F 4 INIT 0 HEAD 'ARRAY' 'ROW'
;
DO WHILE T.ROW LT 6
LET T.ROW = 1 + T.ROW
LET T.TOTGOLD = 0
LET T.TOTSILV = 0
LET T.TOTBRON = 0
CALL PROC TOTAL
CALL PROC PRINT
END
END PROC

*****
* PROC NAME - TOTAL *
* INPUT STREAM PROCEDURE *
*****

TOTAL: PROC TYPE SUBROUTINE, TEMPREINIT
;PERFORM ROW TOTALS
;
;This procedure sums up the number of medals in each category
;(i.e., gold, silver and bronze) for one sport (row).
;
LOC A ROW T.ROW
LET T.TOTGOLD = A.GOLD + T.TOTGOLD
LET T.TOTSILV = A.SILVER + T.TOTSILV
LET T.TOTBRON = A.BRONZE + T.TOTBRON
END PROC

```

Figure E-7 Example 3: ASL Procedures PRNTCNTL and TOTAL

In the ASL procedure PRNTCNTL, the DO WHILE command establishes an explicit loop (back branching). The logical expression is evaluated at the beginning of each iteration. If the value in the temporary field ROW is less than 6, the statements between the DO WHILE command and its corresponding END command are executed. Since T.ROW is set to zero before starting the DO block, the condition on the WHILE operand is initially true and the statements within the DO block are executed. Because T.ROW is incremented within the DO block, the second time through the explicit loop, T.ROW contains a 1; the WHILE condition remains true, allowing the DO block to be executed a second time.


```

(ER) ***** ROW LOOPING DRIVER ***** (E) ( 6) (Y) 25
COMMENT ***** 26
COMMENT ***** 27
COMMENT ***** THIS REQUEST SETS UP A LOOP OF 6 TO CAUSE ALL ROWS OF THE ***** 28
COMMENT ***** ARRAY TO BE PROCESSED ONE ROW AT A TIME. ***** 29
COMMENT ***** 30
-----
STMT FIELD FIELD FLD DEC OUTPT EDIT INITIAL LINE 1 OF LINE 2 OF SEQUENCE LIST
TYPE NAME LNTH TYP PLC EDIT LGTH VALUE COLUMN HEADING COLUMN HEADING 73 TO 80 SEQ
-----
(TF) (COUNTRY) ( 15) (C) (COUNTRY) ) 31
(TF) (SPORT) ( 15) (C) (SPORT) ) 32
(TF) (TOTGOLD) ( 4) (F) (0) (TOTAL) (GOLD) ) 33
(TF) (TOTSILV) ( 4) (F) (0) (TOTAL) (SILVER) ) 34
(TF) (TOTBRON) ( 4) (F) (0) (TOTAL) (BRONZE) ) 35
(TF) (TOTMED) ( 4) (F) (0) (TOTAL) (MEDALS) ) 36
(TF) (GRANDTOT) ( 4) (F) (0) (GRAND) (TOTAL) ) 37
(TF) (PCTGOLD) ( 4) (F) (2) (0) (PCT OF) (GOLD) ) 38
(TF) (PCTSILV) ( 4) (F) (2) (0) (PCT OF) (SILVER) ) 39
(TF) (PCTBRON) ( 4) (F) (2) (0) (PCT OF) (BRONZE) ) 40
(TF) (PCTTOT) ( 4) (F) (2) (0) (PCT OF) (TOTAL) ) 41
(TF) (ROW) ( 4) (F) (0) (ARRAY) (ROW) ) 42
-----
STMT SEQ LOG CON .....OPERAND-A..... OPER .....OPERAND-B..... .....RESULT..... PARTIAL FIELD SEQUENCE LIST
TYPE NO. LEV CIR QLF FIELD SEG-LVL AIN QLF FIELD SEG-LVL (OR CONSTANT) QLF FIELD SEG-LVL STRT NUMB OPR 73 TO 80 SEQ
-----
(PR) (010) (T,ROW ) (LT) (D,6) ) 43
(PR) (020) (T,ROW ) (+) (D,1) (T,ROW ) 44
(PR) (030) (R) (D,0) (T,TOTGOLD) ) 45
(PR) (040) (R) (D,0) (T,TOTSILV) ) 46
(PR) (050) (R) (D,0) (T,TOTBRON) ) 47
(PR) (060) (GO) (SUB TOTAL) ) 48
(PR) (070) (GO) (SUB PRINT) ) 49
(PR) (080) (GO) (REQUEST FRONTNL) ) 50
-----
*****
* REQUEST NAME - TOTAL *
* INPUT STREAM REQUEST *
*****
-----
STMT REPORT REQUESTOR ID MAX SEL SUM VERT FORMS PAGE PAGE LINE REQ BACK BR RNT AB SEQUENCE LIST
TYPE DATE REQUESTOR ID ITEMS CTL RPT SP CNTRL WDIH HGT NOS? TYP SET NAME CONTROL TMP FM 73 TO 80 SEQ
-----
(ER) ***** PERFORM ROW TOTALS ***** (S) (Y) 51
COMMENT ***** 52
COMMENT ***** THIS REQUEST SUMS UP THE NUMBER OF MEDALS IN EACH CATEGORY ***** 53
COMMENT ***** (I.E., GOLD, SILVER AND BRONZE) FOR ONE SPORT (ROW) . ***** 54
-----
STMT SEQ LOG CON .....OPERAND-A..... OPER .....OPERAND-B..... .....RESULT..... PARTIAL FIELD SEQUENCE LIST
TYPE NO. LEV CIR QLF FIELD SEG-LVL AIN QLF FIELD SEG-LVL (OR CONSTANT) QLF FIELD SEG-LVL STRT NUMB OPR 73 TO 80 SEQ
-----
(PR) (010) (A) (LR) (T,ROW ) 56
(PR) (020) (A,GOLD ) (+) (T,TOTGOLD) (T,TOTGOLD) ) 57
(PR) (030) (A,SILVER ) (+) (T,TOTSILV) (T,TOTSILV) ) 58
(PR) (040) (A,BRONZE ) (+) (T,TOTBRON) (T,TOTBRON) ) 59
-----
*****
* REQUEST NAME - PRINT *
* INPUT STREAM REQUEST *
*****
-----
STMT REPORT REQUESTOR ID MAX SEL SUM VERT FORMS PAGE PAGE LINE REQ BACK BR RNT AB SEQUENCE LIST
TYPE DATE REQUESTOR ID ITEMS CTL RPT SP CNTRL WDIH HGT NOS? TYP SET NAME CONTROL TMP FM 73 TO 80 SEQ
-----
(ER) ***** THIS REQUEST COMPUTES THE PERCENTAGES AND PRINTS THE ***** (S) (Y) 60
COMMENT ***** RESULTS ONE SPORT (ROW) AT A TIME. ***** 63
COMMENT ***** 64
-----
STMT SEQ LOG CON .....OPERAND-A..... OPER .....OPERAND-B..... .....RESULT..... PARTIAL FIELD SEQUENCE LIST
TYPE NO. LEV CIR QLF FIELD SEG-LVL AIN QLF FIELD SEG-LVL (OR CONSTANT) QLF FIELD SEG-LVL STRT NUMB OPR 73 TO 80 SEQ
-----
(PR) (010) (A) (LR) (T,ROW ) 65
(PR) (020) (F,ROW ) (TL) (L,NUMSPORT) (T,SPORT) ) 66
(PR) (030) (A,GOLD ) (*) (D,100) (T,PCTGOLD) ) 67
(PR) (040) (T,PCTGOLD) (/) (T,TOTGOLD) (T,PCTGOLD) ) 68
(PR) (050) (A,SILVER ) (*) (D,100) (T,PCTSILV) ) 69
(PR) (060) (T,PCTSILV) (/) (T,TOTSILV) (T,PCTSILV) ) 70
(PR) (070) (A,BRONZE ) (*) (D,100) (T,PCTBRON) ) 71
(PR) (080) (T,PCTBRON) (/) (T,TOTBRON) (T,PCTBRON) ) 72
(PR) (090) (A,GOLD ) (+) (A,SILVER) (T,TOTMED) ) 73
(PR) (100) (T,TOTMED) (+) (A,BRONZE) (T,TOTMED) ) 74
(PR) (110) (T,TOTGOLD) (+) (T,TOTSILV) (T,GRANDTOT) ) 75
(PR) (120) (T,GRANDTOT) (+) (T,TOTBRON) (T,GRANDTOT) ) 76
(PR) (130) (T,TOTMED) (*) (D,100) (T,PCTTOT) ) 77
(PR) (140) (T,PCTTOT) (/) (T,GRANDTOT) (T,PCTTOT) ) 78
(PR) (150) (F,COLUMN) (TL) (L,NUMCNTY) (T,COUNTRY) ) 79

```

Figure E-9 Example 3: Fixed Format Request for Arrays (Page 2 of 2)

Example 4 Reformat Dates

Example 4 shows how easy it is to reformat data using ASL. This example reformats dates.

- Convert MMDDYY to YYMMDD.
- Convert MMDDYY to MM/DD/YY.
- Convert YYMMDD to MM/DD/YY.

[Figure E-10](#) and [Figure E-11](#) show the procedure and request, respectively.

```

*****
*   PROC NAME - DATECONV   *
*   INPUT STREAM PROCEDURE *
*****

DATECONV: PROC TYPE SUBROUTINE
;EXAMPLE OF DATE REFORMATTING
;
YYMMDD:  FIELD C 6
MM_DD_YY: FIELD C 8
;
;Reformat MDDYY to YYMMDD format
COMBINE PF(MDDYY 5 2) MDDYY  STORE T.YYMMDD
;
;Reformat overdefined MDDYY (MONTH DAY YEAR) to MM/DD/YY
COMBINE MONTH '/' DAY '/' YEAR  STORE T.MM_DD_YY
;
;Reformat YYMMDD into MM/DD/YY
COMBINE PF(YYMMDD 3 2) '/' PF(YYMMDD 5 2) '/' PF(YYMMDD 1 2) ,
STORE T.MM_DD_YY
END PROC
    
```

Figure E-10 Example 4: ASL Procedure

```

*****
*   REQUEST NAME - DATECONV *
*   INPUT STREAM REQUEST   *
*****
    
```

SIMT TYPE	REPORT DATE	REQUESTOR ID	MAX ITEMS	SEL CTL	SUM RPT	VERT SP	FORMS CNTRL	PAGE WIDTH	PAGE HIGHT	LINE NOS?	REQ TYP	SET NAME	BACK CONTROL	RR TMP	RNT EM	AB	SEQUENCE 73 TO 80	LIST SEQ
(ER)											(S)							6
COMMENT	*****																	7
COMMENT	*****	EXAMPLE OF DATE REFORMATTING																8
COMMENT	*****																	9

SIMT TYPE	FIELD NAME	FIELD LENGTH	FLD TYP	DEC PLC	OUTPT EDIT	EDIT LGTH	INITIAL VALUE	LINE 1 OF COLUMN HEADING	LINE 2 OF COLUMN HEADING	SEQUENCE 73 TO 80	LIST SEQ
(TF)	(DAYONLY)	(2)	(C)								10
(TF)	(YYMMDD)	(6)	(C)								11
(TF)	(MM_DD_YY)	(8)	(C)								12

SIMT TYPE NO.	SEQ LEV	LOG CTR	CON QLF	OPERAND-A FIELD	OPER SEG-LVL	OPERAND-B FIELD	OPER SEG-LVL	(OR CONSTANT)	RESULT QLF	FIELD SEG-LVL	PARTIAL STRT NUMB	FIELD OPR	SEQUENCE 73 TO 80	LIST SEQ
(PR)	(010)			(MDDYY)	01-1	(C0)	(MDDYY)	01-1	(T,YYMMDD)	(5)	(2)	(A)		13
(PR)	(020)			(C, / /)	(R)	(C, / /)			(T,MM DD YY)					14
(PR)	(020)			(MONTH)	01-1	(R)	(MONTH)	01-1	(T,MM DD YY)	(1)	(2)	(C)		15
(PR)	(020)			(DAY)	01-1	(R)	(DAY)	01-1	(T,MM DD YY)	(4)	(2)	(C)		16
(PR)	(020)			(YEAR)	01-1	(R)	(YEAR)	01-1	(T,MM DD YY)	(7)	(2)	(C)		17
(PR)	(030)			(YYMMDD)	01-1	(R)	(YYMMDD)	01-1	(T,MM DD YY)	(3)	(2)	(B)		18
(PR)	(030)			(YYMMDD)	01-1	(R)	(YYMMDD)	01-1	(T,MM DD YY)	(7)	(2)	(C)		19
(PR)	(030)			(C, / /)	(R)	(C, / /)			(T,MM DD YY)	(3)	(4)	(C)		20
(PR)	(030)			(YYMMDD)	01-1	(R)	(YYMMDD)	01-1	(T,MM DD YY)	(5)	(2)	(B)		21
(PR)	(030)			(T,DAYONLY)	(R)	(T,DAYONLY)			(T,MM DD YY)	(4)	(2)	(C)		22

Figure E-11 Example 4: Fixed Format Request

Conversion Table

Note: Because there may not always be a one-to-one correspondence, the relationships provided here are only approximate.

This appendix relates the fixed format VISION:Builder processing (PR) statement operations to their approximate ASL command.

Operator Type	VISION:Builder Operator	ASL Command
Arithmetic	+, -, *, /	+, -, *, /
Array	LA LC, LD, LR	RELEASE LOCATE
Character Scan	SR, SL, SN	SCAN
Conditional Branching	NS, GS	CASE IF ELSE END
Connectors	OR, AND	Boolean Logical Operators OR, AND, NOT
Data Access Control	RD, RE, RG RS FS	FIND RELEASE FIND
Logic Levels	0 - 9	Conventional Algebraic Notation
Loops and Back Branches		DO UNTIL WHILE FOR FORALL END
Partial Field		PF
Relational	EQ NE LT GT GE LE	EQ or = NE or <> LT or < GT or > GE or ≥ LE or ≤
Replacement	R Cn SS JL, JR	LET COMBINE REPLACE LET with JUSTIFY

Operator Type	VISION:Builder Operator	ASL Command
Table	TL, TN, TB, TS, TI	LOOKUP
Temporary Field Definition	TF	FIELD
Unconditional Branching	GO	CALL CONTINUE GO LEAVE RETURN TRANSFER
Validation	CV, DV	VALIDATE

Symbols

- date delimiter, D-16
- . date delimiter, D-17

Numerics

- 1-9 qualifier, A-3

A

- A qualifier, A-3
- ABEND, D-9
- AINV setting, D-29
- aliases, D-30
- AMIS setting, D-29
- arithmetic, A-3
- arithmetic expressions, 2-6
- ARRAY command, 3-3
- arrays
 - ASTATUS, D-7
 - ROW flag, D-27
- ASA carriage control, 5-56
- ASL
 - description, 2-1, 2-12, 2-13, 2-14, 2-15, 2-16, 2-17, 2-22
 - examples, 7-1
 - implicit loops, 2-12
 - record processing, 2-17
 - segment processing, 2-17

- set operation, 2-12
- terminology, 2-2

- ASTATUS flag, D-7
- audit files, D-13
 - M4AUDIT, D-21
- AVERAGE command, 5-3

B

- B qualifier, A-3
- Blank qualifier, A-3
- books, 1-2
 - IBM Language Environment for MVS & VM Programming Reference, 4-18
 - IBM Language Environment Reference Manual, 4-44
 - VISION:Workbench for DOS Reference Guide, 1-2
- built-in functions, 4-4, 4-5, 4-6, 4-7, 4-8, 4-9, 4-10, 4-11, 4-13, 4-14, 4-15, 4-16, 4-17, A-5
 - conditional, 4-3
 - value, 4-4

C

- CALL command, 4-1, 4-18, 4-20, 4-21, 4-23, 4-24, 4-27, 4-28, 4-29, 4-30, 4-35, 4-36, 4-38, 4-42, 4-43, 4-45, 4-46, 4-47, 4-49, 4-55, 4-56, 4-58
- CALL statement
 - CEEDATE, 4-19
 - CEEDATM, 4-19

CEEDAYS, 4-19
CEEDYWK, 4-19
CEEGMT, 4-19
CEEGMTO, 4-19
CEEISEC, 4-19
CEELOCT, 4-19
CEEQCEN, 4-19
CEESCEN, 4-19
CEESECI, 4-19
CEESECS, 4-19
CEEUTC, 4-19
MOD module-name, 4-19
MODULE module-name, 4-19
PROC procedure-name, 4-18
PROCEDURE procedure-name, 4-18
REP report-name, 4-18
REPORT report-name, 4-18
SUB subfile-name, 4-18
SUBFILE subfile-name, 4-18
USING parm..., 4-20

CALL statements
 CSTATUS flag, D-12
 RETURNCD flag, D-5

CASE command, 4-1, 4-22

CATALOG command, 3-4

CEEDATE, 4-19
CEEDATM, 4-19
CEEDAYS, 4-19
CEEDYWK, 4-19
CEEGMT, 4-19
CEEGMTO, 4-19
CEEISEC, 4-19
CEELOCT, 4-19
CEEQCEN, 4-19
CEESCEN, 4-19
CEESECI, 4-19
CEESECS, 4-19
CEEUTC, 4-19

Chained Coordination
 additional keywords for, 3-23

character, A-2

character constants, 2-3

character date values, 4-19

CHECKPOINT command, 3-6

CHKP flag, D-9

CKPTID flag, D-9

CMS, 1-1, 1-2

COLLATE command, 3-8

COM command, 4-1

COMBINE command, 4-1, 4-23, 4-24, 4-27, 4-28, 4-29, 4-30, 4-35, 4-36, 4-38, 4-42, 4-43, 4-45, 4-46, 4-47, 4-49, 4-55, 4-56, 4-58

COMMAND flag, D-10

command group
 Report, 5-1
 Run Control, 3-1

commands, 4-1
 alphabetic listing of all commands, A-6
 CALL, 4-1
 CASE, 4-1
 COM, 4-1
 COMBINE, 4-1
 CONT, 4-1
 CONTINUE, 4-1
 DO, 4-1
 ELSE, 4-1
 END, 4-1
 FIELD, 4-1
 FLD, 4-1
 GO, 4-1
 IF, 4-1
 LEAVE, 4-2
 LET, 4-2
 LOC, 4-2
 LOCATE, 4-2
 REL, 4-2
 RELEASE, 4-2
 REP, 4-2
 REPLACE, 4-2

RET, 4-2
RETURN, 4-2
TRANSFER, 4-2
comments, 2-6, A-3
COMPUTE command, 5-4
CONDCODE flag, D-11
conditional functions, A-4, A-5
constants, 2-3, 2-4, A-2
CONT command, 4-1
contacting Computer Associates, web page, 1-3
CONTINUE, 2-16
CONTINUE command, 4-1, 4-24
CONTROL command, 3-10
converting character timestamps, 4-19
converts number of seconds, 4-19
coordinated files
 M4CORDn, D-21
 RSTATUS flag, D-29
coordination, D-22
COPY command, 3-18
COUNT command, 5-7
CSTATUS flag, D-12
CUMULATE command, 5-8

D

DATA command, 5-9
database manager, D-10, D-16, D-23, D-25, D-30
DATE, D-12
dates
 ISDATE, D-16
 JULANX, D-17
 JULIAN, D-17
 LILIAN, D-17
 TODAY, D-32
 TODAYX, D-32
day of week, 4-19
DB2, 4-5

SQL flag, D-30
DEBUG command, 3-19
decimal, A-2
decimal constants, 2-4
Definition Processor, 1-1
 VISION:Workbench for ISPF, 1-1
delete, D-2
DELETE flag, D-13
delimiters
 dates, D-16, D-17
direct-read, D-29
DO command, 4-1, 4-25, 4-26, 4-27, 4-29
DOCUMENT command, 3-21
DOS, 1-1
dynamic
 partial field specification, D-18, D-19, D-23, D-27,
 D-28
dynamic report line modification
 diagnostic messages, 5-65

E

E qualifier, A-3
ECORD flag, D-13, D-14, D-22
edit patterns, C-5
EGRP setting, D-29
ELSE command, 4-1, 4-30
END command, 4-1, 4-32, 5-12
EOF flag, D-11, D-15, D-29
examples, 7-1
EXTRACT command
 four variations of, 6-1
EXTRACT DBDNAME command, 6-14
EXTRACT DDNAME command, 6-9
EXTRACT FILE command, 6-1, 6-2
EXTRACT TABLE command, 6-18

F

F qualifier, A-3

FDNAME flag, D-16

FIELD command, 4-1, 4-33, 4-34

field names, 2-4, A-2, C-1

fields

 result, 5-58, 5-60

FILE AUDIT command, 3-22

FILE CORDn command, 3-23

FILE flag, D-16

FILE MASTER command, 3-31, 3-41

 rules, 3-31

FILE MASTER command variation

 variation for in-memory only, 3-31

 variation for input/update, 3-31

 variation for output, 3-31

FILE REJECT command, 3-39

FILE REPn command, 3-40

FILE REPORT command, 3-41

FILE SUBFn command, 3-42

FILE TRAN command, 3-45

FILEID flag, D-16

files

 audit, D-13, D-21

FIND function, 4-5, 4-6, 4-7

flags, D-2

 ASTATUS, D-7

 CHKP, D-9

 CKPTID, D-9

 COLUMN, D-9

 COMMAND, D-10

 CONDCODE, D-11

 CSTATUS, D-12

 DATE, D-12

 DELETE, D-13

 ECORD, D-13, D-14, D-22

 EOF, D-15, D-29

 FDNAME, D-16

 FILE, D-16

 FILEID, D-16

 ISDATE, D-16

 JULANX, D-17

 JULIAN, D-17

 LILIAN, D-17

 LNUMBER, D-18

 LSTART, D-18

 LSTATUS, D-19

 M4AUDIT, D-21

 M4CORDn, D-21

 M4NEW, D-21

 M4OLD, D-21

 M4REJECT, D-22

 M4SUBFn, D-22

 M4TRAN, D-22

 MISSPASS, D-22

 MNUMBER, D-23

 MSTART, D-24

 MSTATUS, D-25

 OWN, D-25

 PAGE, D-26

 PASSWORD, D-26

 RESTART, D-26

 RETURNCD, D-12, D-26

 RNUMBER, D-27

 ROW, D-27

 RSTART, D-28

 RSTATUS, D-29

 SEGNAME, D-30

 SQL, D-30

 SSCOUNT, D-30

 STRAN, D-31

 TIME, D-32

 TODAY, D-32

 TODAYX, D-32

 TRAN, D-33

 XTRAN, D-33

FLD command, 4-1

floating point, A-2

floating point constants, 2-4

FORMAT command, 5-13

G

GDBI, 4-6
GO command, 4-1
GO TO command, 4-36
Greenwich Mean Time (date and time), 4-19
GROUP command, 5-26
GS statements, D-29

H

H qualifier, A-3
half-adjusting, 5-61

I

ICF, D-10
ICOLUMN flag, D-9
ID
 checkpoint ID, D-9
 CHPTID, D-9
 FILEID, D-16
IF command, 4-1, 4-38
implicit loops, 2-12
INCLUDE command, 4-1, 4-40
Indexed Direct Coordination (ICF)
 additional keywords for, 3-23
integer, A-2
integer constants, 2-3
ISDATE flag, D-16
ISPF, 1-1
ITEM command, 5-28

J

J qualifier, A-3
JULANX flag, D-17
JULIAN flag, D-17

K

K qualifier, A-3
KEQL setting, D-29
key field, D-33, D-34
keywords
 DECIMALEEDIT, 5-57, 5-62
 NOCALC, 5-57
 NOCALC POSnnn, 5-57, 5-62
 NOSPACELINES, 5-56
 number of occurrences, 5-64
 override, 5-64
 PERCENT, 5-57, 5-58
 RATIO, 5-57, 5-60
 ROUND, 5-57, 5-61, 5-62
 SKIP c POSnnn, 5-56
 SPACEAFTER, 5-56
 SPACEBEFORE, 5-56
 SUPPRESS SPACE, 5-57, 5-63
 TITLE LINENO, 5-55, 5-62
KGTR setting, D-29

L

labels, 2-4
LEAVE, 2-16
LEAVE command, 4-2, 4-42
LET command, 4-2, 4-44, 4-45, 4-46
Lilian, 4-19
LILIAN flag, D-17
LINE command, 5-31
LINKAGE command, 3-47
list expressions, A-4
LISTCNTL command, 3-48
LISTLIB GLOSSARY command, 3-50
LISTLIB NAMES command, 3-51
LNUMBER flag, D-18
LOC command, 4-2
LOCATE command, 4-2, 4-48, 4-49

LOCATE function, 4-7
Logical expressions, 2-7, 2-8
logical expressions, A-4
LOOKUP function, 4-8, 4-9, 4-10
LSTART flag, D-18
LSTATUS flag, D-19

M

M qualifier, A-3
M4AUDIT flag, D-21
M4CORDn flags, D-21
M4NEW, D-21
M4NEW flag, D-21
M4OLD flag, D-21
M4REJECT, D-22
M4REJECT flag, D-22
M4TRAN flag, D-22
mapping requests
 PASSWORD, D-26
master files, D-31, D-33
 M4NEW, D-21
 M4OLD, D-21
 MISSPASS, D-22
 New, D-21
 Old, D-21
MAX command, 5-34
MIN command, 5-35
MISSPASS flag, D-22
MNUMBER flag, D-23
MODULE keyword, 4-18
MODULE module-name, 4-19
MOSAIC, 4-6
MSTART flag, D-24
MSTATUS flag, D-25
MSUBFn flags, D-22
MULTILIB command, 3-52

MVS, 1-1

N

N qualifier, A-3
names, 2-4
NEWPAGE command, 5-36
NREC setting, D-29
NS statements, D-29

O

O qualifier, A-3
ORDER command, 5-37
output edit, C-9, C-10, C-11, C-12
OVERRIDE command, 3-53
overrides
 keywords, 5-64
OWN flag, D-25
own-code
 OWN flag, D-25
OWNCODE command, 3-54

P

PAGE flag, D-26
page layout, 2-11
Parallel Looping, 6-11, 6-20
partial fielding, D-14
 dynamic, D-18, D-19, D-23, D-27, D-28
 dynamic specification, D-27
 errors, D-27
 fixed point text, D-18, D-27
 LNUMBER, D-18
 MNUMBER, D-23
 MSTART, D-24
 RNUMBER, D-27
 RSTART, D-28
 standard specification, D-27
 text processing, D-18, D-24, D-27, D-28

PASSWORD flag, D-26
pattern, A-2
patterns, 2-4, C-4
PERCENT command, 5-38
PF function, 4-11
PINV, D-12
PMIS, D-12
PREFACE command, 5-40
PROC command, 4-2, 4-50
PROC procedure-name, 4-18
PROCEDURE procedure-name, 4-18
procedures/requests
 type 1, D-13
 type 2, D-13
 type 3, D-13
 type M, D-13
 type N, D-13

Q

Q qualifier, A-3
qualifiers, 2-5, A-3, C-4
 F flag, D-2

R

RATIO command, 5-41
record processing, 2-17
reject files, D-22
REL command, 4-2
relational
 expressions, A-4
RELEASE command, 4-2, 4-55, 4-56
REP command, 4-2
REP report-name, 4-18
REPLACE, D-30
REPLACE command, 4-2, 4-57, 4-58
report block, 5-1

REPORT command, 5-42
Report command group, 5-1
REPORT report-name, 4-18
reports, D-16
request types, D-21, D-22, D-31
 P preselection, D-9
request-read, D-29
requests
 mapping, D-25
 preselection, D-9
reserved words, C-1
RESTART flag, D-26
result field, 5-58
RET command, 4-2
RETRIEVE command, 3-55
retrieves current date, 4-19
retrieves current time, 4-19
return codes, D-26
RETURN command, 4-2, 4-59
RETURNCD flag, D-12, D-26
RIGN setting, D-29
RNUMBER flag, D-27
rounding, 5-61
ROUTE command, 3-57
ROW flag, D-27
RPCORDONLY, D-22
RSTART flag, D-28
RSTATUS flag, D-29
rules regarding statements allowed for various
 types of runs, 3-2
Run Control command group, 3-1

S

SCAN function, 4-13, 4-14, 4-15
SECTION command, 5-50
segment names

SEGNAME flag, D-30
segment processing, 2-17
segments, D-9, D-31
 LSTATUS, D-19
SEGNAME flag, D-30
set operation, 2-12
setting the century window, 4-19
SIZE command, 5-51
SKIP command, 5-52
SPACEAFTER POSnnn x#, 5-64
SQL flag, D-30
SSCOUNT flag, D-30
Standard Coordination
 additional keywords for, 3-23
statement labels, A-2
Statement syntax, 2-10
statement syntax, A-4
STRAN flag, D-31
SUBFILE subfile-name, 4-18
subfiles
 M4SUBFn, D-22
suppress
 blank lines, 5-56
 spaces, 5-63
syntax of functions and commands, 2-10
system defined names, C-1

T

T qualifier, A-3
technical support, contacting Computer Associates,
1-3
terminology, 2-2
time, A-2
time constants, 2-4
TIME flag, D-32
timestamp, 4-19

TITLE command, 5-53
titles, D-26
TODAY flag, D-32
TODAYX flag, D-32
TOTAL command, 5-54
TRACK command, 3-59
TRAN flag, D-33
transaction file, D-33
transactions
 M4TRAN file, D-22
 STRAN flag, D-31
TRANSFER command, 4-2, 4-60

U

update-in-place, D-21
User Read files
 additional keywords for, 3-23
USING parm..., 4-20

V

V qualifier, A-3
VALIDATE function, 4-16, 4-17
validation patterns, C-4
value functions, A-5
variable length fields, D-30
VISION:Builder, 1-1
VISION:Inform, 1-1
VISION:Two, 1-1
VISION:Workbench for DOS, 1-1
VISION:Workbench for ISPF, 1-1
 aka Definition Processor, 1-1
VSE, 1-2

W

W qualifier, A-3

web page

 Computer Associates, 1-3

WORK command, 3-61

X

X qualifier, A-3

XREP command, 5-55

XTRAN flag, D-33

