Advantage™ CA-Easytrieve®

Language Reference Guide

Computer Associates®

Contents

Chapter 1: Overview

Chapter 2: Statement Summaries

Chapter 3: Statements A - C

Chapter 4: Statements D - F

Chapter 5: Statements G - M

Chapter 6: Statements N - R

Chapter 7: Statements S - Z

Appendix A: System-Defined Files

Appendix B: Symbols and Reserved Words

Appendix C: Conversion from CA-Easytrieve Plus (Batch)

Index

Chapter

1 Overview

Introduction

CA-Easytrieve is an information retrieval and data management system designed to simplify computer programming. Its English-like language and simple declarative statements provide the new user with the tools needed to produce comprehensive reports and screens with ease, while its enhanced facilities provide the experienced data processor with the capabilities to perform complex programming tasks.

CA-Easytrieve operates on the IBM 370, 30xx, 43xx, and compatible processors in the VM, MVS, and VSE environments. Under TSO, CMS, and CICS, CA-Easytrieve runs interactively for data inquiry, analysis, and reporting. The output can be either returned to your terminal screen or routed to a printer.

CA-Easytrieve/Workstation operates on the IBM/PC (or 100 percent compatible) in the PC/DOS and OS/2 environments.

CA-Easytrieve also operates on the HP-9000 Series 700/800 in the HP-UX environment, and on the IBM RS/6000 in the AIX environment.

About This Guide

The *CA-Easytrieve Language Reference Guide* is your source for complete language details. It contains descriptions of the complete syntax of each CA-Easytrieve statement, organized in easy-to-find alphabetical order. It also provides:

- A list of system-defined fields.
- A list of CA-Easytrieve symbols and reserved words.
- Information for those converting to this version of CA-Easytrieve.

Note: You should review Appendix C, "Conversion from CA-Easytrieve Plus (Batch)," if you are a current user of CA-Easytrieve Plus (batch environment).

This guide is to be used by programmers using the following implementations of CA-Easytrieve:

- CA-Easytrieve/Online, version 1.4
- CA-Easytrieve/Workstation, version 1.3
- CA-Easytrieve in the UNIX environment, version 1.4.

Organization

This guide is divided into seven chapters and three appendices:

Chapter 1, Overview Introduces you to CA-Easytrieve and the syntax rules for coding a CA-Easytrieve program.

Chapter 2, Statement Summaries Provides two types of quick reference summaries:

- Alphabetical Statement Summary - an alphabetical list of all statements and an overview of the usage of each statement.
- Functional Category Summary - a list of the functional categories of statements, the statements in each function, and their usage.

Chapter 3, Statements A - C Provides the complete syntax of statements beginning with A through C.

Chapter 4, Statements D - F Provides the complete syntax of statements beginning with D through F.

Chapter 5, Statements G - M Provides the complete syntax of statements beginning with G through M.

Chapter 6, Statements N - R Provides the complete syntax of statements beginning with N through R.

Chapter 7, Statements S - Z Provides the complete syntax of statements beginning with S through Z.

Appendix A, System-Defined Fields Contains a list of CA-Easytrieve system-defined fields.

Appendix B, Symbols and Reserved Words Contains a list of CA-Easytrieve symbols and reserved words.

Appendix C, Conversion from CA-Easytrieve Plus (Batch) Contains information for converting to this implementation of CA-Easytrieve from prior implementations.

Index Provides a quick way to find references to functions, procedures, and terms.

Related Publications

The following publications, produced by Computer Associates, are either referenced in this publication or are recommended reading:

- *CA-Pan/SQL SQL Interface Installation Guide*
- *CA-PSI Subsystems DBCS Environment Guide*
- *CA-PSI Subsystems Reporting Environment Guide.*

The following publications, not produced by Computer Associates, are either referenced in this publication or are recommended reading:

- IBM IMS/DL/I Application Programming publications
- *BTRIEVE Reference Manual*
- *CICS Programmer's Reference Manual*.

Documentation Conventions

The following conventions are used throughout this guide for illustrative purposes:

Notation	Meaning
{braces}	Mandatory choice of one of these entries.
[brackets]	Optional entry or choice of one of these entries.
| (OR bar)	Choice of one of these entries.
(parentheses)	Multiple parameters must be enclosed in parentheses.
...	Ellipses indicate that you can code the immediately preceding parameters multiple times.
BOLD	Bold text in program code is used to highlight an example of the use of a statement.
CAPS	All capital letters indicate a CA-Easytrieve keyword, or within text descriptions, indicate a name or field used in a program example.
lowercase italics	Lowercase italics represent variable information in statement

Notation	Meaning
	syntax.

Summary of Revisions

The following lists summarize the technical changes and enhancements provided in version upgrades of CA-Easytrieve.

CA-Easytrieve/Online Enhancements from CA-Easytrieve Plus

- Full 31-bit addressing in MVS environments.
- Environment-independent FILE statements ensure portability between environments and access methods.
- The CLOSE statement now allows controlled file opens and closes.
- A dynamic file name provides the ability to determine the file name at execution time.
- Simple read/write access to SQL files provides automated cursor management with full application capabilities
- Complete control over SQL units of work using the COMMIT statement and activity options.
- 128-character entity names for ANSI standard support.
- Use of descriptive logical file names greater than 8 characters.
- Boundary checking of subscripts and indices during execution protects environment and makes debugging easier.
- Introduction of the PROGRAM “super” activity that can execute other activities as logic dictates.
- Direct access to execution parameters through the PROGRAM statement.
- Ability to LINK and TRANSFER to other CA-Easytrieve programs.
- SCREEN activities provide easy creation of online transaction processing applications. Screen generation and maintenance assisted by Screen Painter.
- SEARCH of INDEXED table file results in keyed read rather than binary search.
- Online Report Display Facility allows browsing of report output.
- Access to report line and page counters.
- Ability to modify report lines in BEFORE-LINE procedures.

- Ability to specify column locations for title items in automatically adjusted reports.
- Fully integrated support of DISPLAY statements in REPORTs with page-break handling and consistent production of titles and headings.
- Integrated syntax-directed editor and interpreter giving compatible development tools and rapid prototyping abilities.
- Source program input directly from CA-Panvalet and PIELIB libraries.
- Access to multiple macro library types during compilation.
- Issuance of warning messages during compilation provide helpful direction.
- Enhanced compilation listing.
- Report Painter provides visual environment for creation and maintenance of report declarations.

CA-Easytrieve/Workstation Enhancements from CA-Easytrieve Plus/PC

- Compatibility with mainframe implementations including portable file and field definition, arithmetic functions, reporting, and screen handling.
- Generation of EXE modules for significant performance improvement and decreased memory requirements.
- Full generation of Intel object code for compatibility with other PC compilers.
- Environment-independent FILE statements ensure portability between environments and access methods.
- Direct program access to CA-IDMS, CA-Datacom, dBASE, LOTUS, CA-SuperCalc, and comma-delimited files.
- Complete control over SQL and CA-IDMS units of work using the COMMIT statement and activity options.
- Simple read/write access to SQL files.
- CLOSE statement allows controlled file opens and closes.
- Dynamic file name provides ability to determine file name at execution time.
- 128-character entity names for ANSI standard support.
- Use of descriptive logical file names greater than 8 characters.
- Subscripting of arrays.
- Boundary checking of subscripts and indices during execution protects environment and makes debugging easier.
- Introduction of the PROGRAM “super” activity that can execute other activities as logic dictates.

- Direct access to execution parameters through the PROGRAM statement.
- Ability to LINK and TRANSFER to other CA-Easytrieve programs as well as to any operating system command using a command shell.
- Direct CALLs to subroutines written in C, Assembler, and COBOL.
- SCREEN activities provide easy creation of online transaction processing applications.
- SEARCH of INDEXED table file results in keyed read rather than binary search.
- Report Display Facility allows browsing of report output.
- Access to report line and page counters.
- Ability to modify report lines in BEFORE-LINE procedures.
- Ability to specify column locations for title items in automatically adjusted reports.
- Fully integrated support of DISPLAY statements in REPORTs with page-break handling.
- Enhanced compilation listing.
- GRAPH subactivities provide business graphics with the ease of CA-Easytrieve reporting.
- HLLAPI Host Interface providing automated function shipping to the mainframe and behind-the-scenes file transfers.

CA-Easytrieve/Online 1.1 Enhancements

Windowed Screens	SCREENs can now be any size and have any start location. When a SCREEN activity executes another SCREEN activity in which the second screen is smaller than the first, the second screen overlays the first as a “pop-up” window. Screens can now have a border built displayed around them, whether they are full-screen applications or windows.
SET Statement	The SET statement provides an easy method to change screen attributes for a field or to indicate an erroneous field in your screen procedures.
SCREEN Attributes	The CURSOR attribute has been added to the default set of attributes applied to fields in error. Existing users may want to add the attribute in their site options.
GET PRIOR Statement	The PRIOR parameter on the GET statement allows backwards browses against VSAM files. In addition, you can load a

CA-Easytrieve virtual file and browse forward and backward through the file.

Working Storage Reinitialization (DEFINE RESET)

A RESET parameter on the DEFINE statement allows you to specify that W working storage fields are initialized automatically for each execution of a JOB, SORT, or SCREEN statement.

INTEGER/ROUNDED/ TRUNCATED on Assignment Statement

These options provide the following capabilities:

- Automatic dropping of fractional results of calculations or assigns.
- Automatic rounding off of fractional results of calculations
- Truncation of digits during an assignment.

The INTEGER parameter can be used with ROUNDED or TRUNCATED. Additional calculations and multiple Assignment statements previously required to perform these functions are no longer needed.

Instream Macros

The compiler now supports including macro definition as part of the source program. This capability is particularly useful for testing new macros prior to storing them in the macro library.

Multiple Name Support

CA-Easytrieve/Online now supports multiple entities with the same name. For example JOBs, FILEs, fields, keywords can all have the same name.

TRANSFER NOCLEAR

The NOCLEAR parameter on the TRANSFER statement tells CA-Easytrieve to leave the screen displayed as it terminates the program and transfers control to another program. The terminal user is able to still see the screen display as the target program processes and is not left with a blank screen.

Double Byte Character Set Support

CA-Easytrieve/Online now supports the IBM Double Byte Character Set (DBCS). Kanji and mixed fields and literals can be displayed on and received from the terminal.

Report Painter

A Report Painter provides a visual method for creating and maintaining report declarations. Screens and reports can be painted online using the same easy-to-use interface.

Screen Painter Enhancements

A new **Field Select** window is available to display program fields for selection from other windows and lists.

The **Repeat Definition** panel now automatically generates subscript fields on ROW statements.

A new **CAPS** command provides a session override of the CA-Easytrieve/ESP Editor **CAPS** setting.

CASE Statement	The CASE statement now supports variable length fields. If *field-name* is an alphanumeric literal, it no longer must be 254 or fewer bytes in length.

CA-Easytrieve/Online 1.4 Enhancements

Extended Reporting	CA-Easytrieve/Online now uses the CA-PSI Subsystems Reporting Environment to generate printer set definitions. The Reporting Environment provides support for Impact Dot, Ink-Jet, and Electro-Photographic printers. This facility interacts with CA-Easytrieve report processing to provide support for many additional features, such as font control. These are described fully in the *CA-Easytrieve Programmer Guide*.
Synchronized File Processing	The Synchronized File Processing (SFP) facility is now available in CA-Easytrieve/Online.
File Exits	You can now use the EXIT parameter of the FILE statement to invoke subprograms written in other programming languages for input/output related events.
Label Reports	Label reports are now available in CA-Easytrieve/Online.
Even Precision for Packed Fields	You can use the EVEN parameter on the DEFINE statement to indicate that a packed decimal field is to contain an even number of digits.
MOVE LIKE Statement for Working Storage	The MOVE LIKE statement now supports moving contents of fields with identical names to and from working storage.
Static Call Support for Subroutines	The CALL parameter is now available on the PARM statement. CALL enables you to specify how subprograms referenced in CALL statements are linked to your CA-Easytrieve program. The DECLARE statement specifies how a particular subprogram is linked and overrides the CALL parameter on the PARM statement.
IF BREAK	New IF BREAK/HIGHEST-BREAK class tests can be used as alternatives in testing report control breaks.
CONTROLSKIP	The CONTROLSKIP parameter is available on the REPORT statement. CONTROLSKIP enables you to define the number of blank lines to be inserted following CONTROL total lines and the next detail line.
Identifiers and DBCS	Identifiers can now contain DBCS characters.

Year 2000 Support	A SYSDATE-LONG field is now available that contains the century. SHORTDATE and LONGDATE options have been added to the REPORT statement to display the date with or without the century. A new Options Table entry has been added called LONGDTE. This specifies the default date for reports.

CA-Easytrieve/Workstation 1.2 Enhancements

CA-IDMS Processing (PC/DOS Only)	The CA-Easytrieve interface to CA-IDMS is now available in CA-Easytrieve/Workstation. This interface provides complete facilities for information retrieval and maintenance of CA-IDMS/PC databases.
SQL Processing (PC/DOS Only)	The CA-Easytrieve SQL interface is now available in CA-Easytrieve/Workstation. This interface provides complete facilities for information retrieval and maintenance of SQL tables. SQL processing for Version 1.2 supports CA-Datacom/PC.
MOVE LIKE Statement for Working Storage	The MOVE LIKE statement now supports moving contents of fields with identical names to and from working storage.
OS/2 Support	CA-Easytrieve/Workstation now supports OS/2 2.0 and above. This support includes development and execution of full-screen and windowed text-based applications. However, there is no database support.
ASCII Numeric Data	CA-Easytrieve/Workstation now supports ASCII data in N type fields.
Distributable Applications	Your CA-Easytrieve/Workstation applications are fully distributable. If you use the Report Display Facility, you must distribute the EZBR.EXE file with your CA-Easytrieve/Workstation applications. If you use the Graph Display Facility, you must distribute the EZGR.EXE file with your CA-Easytrieve/Workstation applications.

CA-Easytrieve/Workstation 1.3 Enhancements

Year 2000 Compliant	Options and system-defined fields let users specify a 4-digit year on report dates and system dates.

Enhanced CA-Easytrieve/ Workstation Editor	Source files can exceed a file size of 32,768 bytes. A customizable Toolbar incorporates frequently used editor functions into point-and-click icons. Program text is color-coded to offset CA-Easytrieve keywords, alpha character strings, and programmer-defined field names and labels.
CA-Realia File System Interface	You can write CA-Easytrieve programs to create, read, write, and updates CA-Realia native-format files with the CA-Realia file system interface available in CA-Easytrieve Workstation Release 1.3.

CA-Easytrieve 1.3 for UNIX Enhancements

CA-Easytrieve is now available for use in the HP-UX environment and operates on the HP-9000 Series 700/800.

CA-Ingres Processing	The CA-Easytrieve SQL Interface now supports CA-Ingres in the UNIX environment.
Oracle Processing	The CA-Easytrieve SQL Interface now supports Oracle in the UNIX environment.
C-ISAM Processing	CA-Easytrieve now supports C-ISAM files as INDEXED file types.
Year 2000 Support	A SYSDATE-LONG field is now available that contains the century. SHORTDATE and LONGDATE options have been added to the REPORT statement to display the date with or without the century.

A new Options Table entry has been added called LONGDTE. This specifies the default date for reports.

CA-Easytrieve 1.4 for UNIX Enhancements

CA-Easytrieve now operates on the IBM RS/6000 in AIX.

DB2 Processing	The CA-Easytrieve SQL Interface now supports DB2 in the UNIX environment.
SYBASE Processing	The CA-Easytrieve SQL Interface now supports SYBASE in the UNIX environment.
Extended Reporting	A subset of the Extended Reporting feature is now available in UNIX. See the *CA-Easytrieve for UNIX User Guide* for details.
Tool Kit Routines	A subset of CA-Easytrieve Tool Kit is now available in UNIX. See the *CA-Easytrieve Tool Kit User Guide* for details.

Syntax Rules

The free-form English language structure of CA-Easytrieve makes it easy for you to develop an efficient, flexible programming style. To avoid programming errors, follow the simple syntax rules of CA-Easytrieve.

Statement Area

All source statements are records of 80 characters each. A system installation option establishes a statement area within the 80 available positions. The default statement area is in columns 1 through 72.

For example, although positions 1 to 80 are available, 'SCANCOLS 7, SCANCOLE 72' establishes the statement area as positions 7 to 72. This allows for optional data (for example, sequence numbers and program identifiers) to be entered on the record, but still be ignored by CA-Easytrieve. The complete record is always printed on the statement listing.

```
                                         7 7      8
1....6 7.................................2 3......0

001000                                     PRGMNAME

ignored          statement area              ignored
```

Character Sets

CA-Easytrieve/Online supports both Single Byte Character Sets (SBCS) and Double Byte Character Sets (DBCS). MIXED format is data that contains both SBCS and DBCS formatted data.

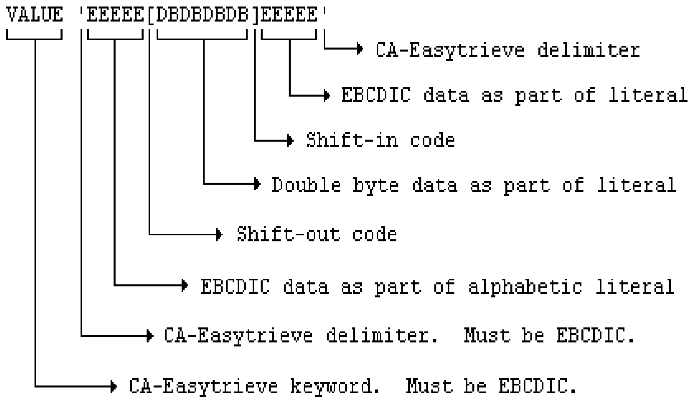
Note: The current release of CA-Easytrieve/Online limits support for alternate character sets to the IBM (Japanese) character set. Report processing is further limited to EBCDIC and MIXED data formats for standard reports and to EBCDIC and DBCS data formats for extended reports. Identifiers (fields names, labels, and so on) can contain DBCS and MIXED data as part of the name. MIXED data format is supported only to the extent that data is allowed. CA-Easytrieve provides no additional formatting based on the alignment of the DBCS portion of the data. DBCS data is not supported on the workstation or in UNIX.

Both EBCDIC and DBCS data processing are applicable to China (Hanzi characters), Japan (Kanji, Hiragana, and Katakana characters), and Korea (Haja and Hangual characters). CA-Easytrieve supports both character sets based upon the following assumptions and rules:

1. All the syntax rules described in this chapter apply to EBCDIC data only. DBCS data in the CA-Easytrieve statement area is not processed for continuation characters, delimiters, words, identifiers, and so on.

2. A DBCS character occupies two bytes in storage. If not identified as DBCS characters, these same two bytes would be processed as a pair of single byte EBCDIC characters. In order to distinguish EBCDIC data from DBCS data, CA-Easytrieve uses a shift code system. This system, called the **Wrapping shift code system**, takes the form of two codes — one code preceding and the second following the DBCS data. These codes wrap or enclose the DBCS data, thereby identifying the beginning and end of DBCS data. The term associated with the code that precedes the DBCS data is a **Shift-Out** code (shift-out of EBCDIC). The code that delimits (separates) the DBCS data is called a **Shift-In** code (shift-in to EBCDIC). These codes can be one or two bytes in length.

   The following illustrates the use of the Wrapping shift code system:


VALUE 'EEEEE[DBDBDBDB]EEEEE'
CA-Easytrieve delimiter
EBCDIC data as part of literal
Shift-in code
Double byte data as part of literal
Shift-out code
EBCDIC data as part of alphabetic literal
CA-Easytrieve delimiter. Must be EBCDIC.
CA-Easytrieve keyword. Must be EBCDIC.

3. A shift code is a special one or two byte character contained in the CA-Easytrieve statement area. Shift code values are defined in the CA-PSI Subsystems DBCS Options module (see the *CA-PSI Subsystems DBCS Environment Guide* for more details on the Options module). Each shift code value uniquely identifies the DBCS code system of the data. If the system cannot be uniquely identified, a default is assumed. You can alter this default at compile time using the PARM statement. See PARM Statement in the "Statements N–R" chapter for more information.

4. In the statement area, CA-Easytrieve requires shift codes to distinguish DBCS data from EBCDIC data. Once a CA-Easytrieve word has been identified, the word is known to be of EBCDIC, DBCS, or MIXED data format. CA-Easytrieve only maintains shift codes for MIXED words. The CA-Easytrieve compiler identifies the statement containing the word and when necessary, performs the required processing to remove the shift codes and convert EBCDIC data.

5. Once CA-Easytrieve finds a shift-out code in a word, the data that follows will not be processed as DBCS data when:
    - The end statement area is reached before the related shift-in code is found.
    - The shift-in code is found but it is not on a double byte boundary.
    - The shift-out code is found as part of the identified DBCS data.

Multiple Statements

The statement area normally contains a single statement. However, you can enter multiple statements on a single record. The character string '. ' (period followed by a space) indicates the end of a statement. Another CA-Easytrieve statement begins at the next available position of the statement area (after the space). For example, the following two CA-Easytrieve statements are on one record:

```
COST = FIXED + VARIABLE.  PRICE = COST + PROFIT
```

Comments

When the first non-blank character of a statement is an '*' (asterisk), the remainder of that record is a comment statement. (It is ignored by the CA-Easytrieve compiler.) You can use comment statements at any place within a program, except within a continued statement. A statement containing all blanks is treated as a comment.

To place a comment on the same line as a statement, code a period (.), one or more spaces, an asterisk (*), and then the comment.

Continuations

The last non-blank character of a statement terminates the statement unless that character is a - (minus) or a + (plus). The - indicates that the statement continues at the start of the next statement area. The + indicates that the statement continues with the first non-blank character in the next statement area. The difference between - and + is important only when continuing words. Continuation between words is the same for both. The following continued statements produce identical results:

```
FIELD-NAME   W   6   A   +
             VALUE   'ABC-
DEF'
----------------------------
FIELD-NAME   W   6   A   -
             VALUE   'ABC+
                      DEF'
```

DBCS Data

To continue a statement defining DBCS data, you must delimit the DBCS data. This means a shift-in code must precede the continuation character and a shift-out code must precede the continuing DBCS data on the next record. The following example illustrates continuing a DBCS literal:

```
FIELD-NAME   W  10   K   +
             VALUE   '[DBDBDB]   +
[DBDB]'
```

The [and] indicate shift-out and shift-in codes.

Words and Delimiters

One or more words make up each CA-Easytrieve statement. A word can be a keyword, field name, literal, or symbol. All words begin with a non-blank character. A delimiter or the end of the statement area terminates these words. Delimiters make statements readable but are not considered part of the attached word. CA-Easytrieve word delimiters are:

Delimiter	Description
space	The basic delimiter within each statement.
‘ single quote	Encloses literals which are alphanumeric.
. period followed by a space	Terminates a statement.
, comma	Used optionally for readability.
() parentheses	Encloses multiple parameters and portions of arithmetic expressions (the left parenthesis acts as a basic delimiter).
: colon	Used as a delimiter for file, record, and field qualifications.

At least one space must follow all delimiters except for the ‘(‘ (left parenthesis) and ‘:’ (colon). The word RECORD-COUNT is shown below with various delimiters:

```
RECORD-COUNT
FILEONE:RECORD-COUNT
(RECORD-COUNT)
‘RECORD-COUNT'
RECORD-COUNT,
RECORD-COUNT.
```

Keywords

Keywords are words that have specific meaning to CA-Easytrieve. Some keywords are reserved words. You can use non-reserved keywords in the appropriate context as field names, whereas reserved words cannot be used as field names. For more information on keywords and reserved words, see Appendix B.

Multiple Parameters

You must enclose multiple parameters within parentheses to indicate group relationships. If parentheses are not used, only one parameter is assumed. The following example is a CA-Easytrieve statement with multiple parameters:

```
CALL   PGMNAME  USING(FIELDA, FIELDB, FIELDC)
```

Field Names

Field names are composed of a combination of not more than 128 characters chosen from the following:

- Alphabetic characters, A through Z, lowercase as well as uppercase
- Decimal digits 0 through 9
- All special characters, except delimiters.

The first character of a field name must be an alphabetic character, a decimal digit, or a national character (#, @, $). In addition, a field name must contain at least one alphabetic or special character to distinguish the field name from a number.

All working storage field names must be unique, as well as all field names within a single file. If you use the same field name in more than one file, or in a file and in working storage, you must qualify the field name with the file name or the word WORK. A qualified field name consists of the qualifying word followed by a colon and the field name. You can use any number of spaces, or no spaces, to separate the colon from either the qualifying word or the field name. Field names can contain DBCS characters.

Assume FLD1 occurs in both working storage and the file FILEA. FLD1 can be qualified in the following ways:

```
FILEA: FLD1
FILEA:FLD1
FILEA : FLD1
WORK:FLD1
```

Labels

Labels identify specific PROGRAMs, JOBs, PROCedures, REPORTs, SCREENs and statements. Labels can be 128 characters long, can contain any character other than a delimiter, and can begin with A through Z, 0 through 9, or a national character (#, @, $); they cannot consist of all numeric characters. Labels can contain DBCS characters.

Identifiers

Identifiers are words which name things (field names, statement labels, etc.) in CA-Easytrieve. Identifiers cannot contain these delimiters:

```
,  comma
'  single quote
(  left parenthesis
)  right parenthesis
:  colon
```

Arithmetic Operators

CA-Easytrieve arithmetic expressions use the following arithmetic operators:

```
*  multiplication
/  division
+  addition
-  subtraction
```

The arithmetic operator must lie between two spaces.

Numeric Literals

Numeric literals can contain 18-numeric digits (characters 0 through 9). You can indicate the algebraic sign of a numeric literal by attaching a + (plus) or a - (minus) prefix to the numeral. Also, you can use a single decimal point to indicate a maximum precision of up to 18 decimal positions. The following examples are valid numeric literals:

```
123
+123
-123.4321
```

Alphanumeric Literals

Alphanumeric literals are words enclosed within single quotes, and can be 254 characters long. See Literal and Data Formatting Rules, in the “Coding a CA-Easytrieve Program” chapter of the *CA-Easytrieve Programmer Guide*, for complete information on literals.

Font Numbers

Font numbers are used by extended report processing to identify which font is used to display the field name or literal. A font number must begin with a pound sign (#) and contain only numeric digits. Font numbers can be specified on the following statements:

- DEFINE
- DISPLAY
- HEADING
- LINE
- TITLE

Chapter

2

Statement Summaries

Introduction

This chapter provides a quick reference summary of all CA-Easytrieve statements. Two types of cross-references are provided:

- Alphabetical Statement Summary - an alphabetical list of all statements and an overview of the usage of each statement.
- Functional Category Summary - a list of the functional categories of statements, the statements in each function, and their usage.

Alphabetical Statement Summary

Statement	**Usage**
%	Invoke a macro
*	Document comments in a program
ACCESS	Access a macro secured against unauthorized access in CA-PANVALET or VSAM
AFTER-BREAK	A REPORT procedure invoked following the printing of summary lines for a control break
AFTER-LINE	A REPORT procedure invoked after printing a detail line on a report
AFTER-SCREEN	A SCREEN procedure performed after a SCREEN activity receives data from the terminal
Assignment	Establish a value in a field
BEFORE-BREAK	A REPORT procedure invoked before printing the summary lines for a control break
BEFORE-LINE	A REPORT procedure invoked before printing a detail line on a report
BEFORE-SCREEN	A SCREEN procedure invoked before a SCREEN activity sends data to

Statement	Usage
	the terminal
CALL	Invoke subprograms written in other programming languages
CASE	Conditionally execute one of several alternative groups of statements based on the value of a specific field
CLOSE	Close a file
COMMIT	Commit a logical unit of recoverable work
CONTROL	Identify control fields used in a control report
COPY	Duplicate field definitions of a named file
CURSOR	Set the initial position of the screen cursor
DECLARE	Name a set of screen attributes or an input edit pattern, or whether a program is statically or dynamically linked
DEFAULT	Override system-defined screen attributes and message locations
DEFINE	Specify a data field within a file or within working storage
DELETE	Delete a row from a CA-Easytrieve SQL file
DISPLAY	Format and transfer data to the system output device or to a named file
DLI	Perform IMS/DLI functions against an IMS/DLI database.
DO UNTIL	Control repetitive program logic by evaluating the condition at the bottom of a group of statements
DO WHILE	Control repetitive program logic by evaluating the condition at the top of a group of statements
DRAW	Produce graphic output by initiating a GRAPH subactivity
ELEMENT-RECORD	Identify the element records that comprise the logical record
ELSE	Identify statements to be executed when IF conditions are false. See IF.
ELSE-IF	Identify a conditional expression to be tested when the previous IF or ELSE-IF conditional expression is false. See IF.
END-CASE	Terminate the body of a CASE statement. See CASE.
END-DO	Terminate the body of a loop associated with a DO UNTIL or DO WHILE statement. See DO UNTIL and DO WHILE.
END-IF	Terminate the logic associated with the previous IF statement. See IF.
ENDPAGE	A REPORT procedure used to produce page footing information
END-PROC	Delimit the statements in a procedure. See PROC.
END-REPEAT	Terminate the body of a REPEAT statement. See REPEAT.

Statement	Usage
ENDTABLE	Delimit instream data used to create small tables
EXECUTE	Invoke a JOB, SORT, or SCREEN activity from a PROGRAM or SCREEN activity
EXIT	Terminate a SCREEN activity
FETCH	Retrieve a row from a CA-Easytrieve SQL file
FILE	Describe a file and database references
GET	Place the next sequential record of the named file into the file's record buffer
GOTO	Modify the top to bottom logic flow of statement execution
GOTO JOB	Branch to the top of the current JOB activity
GOTO SCREEN	Branch to the top of the current SCREEN activity
GRAPH	Define the style and characteristics of a graph
HEADING	Define an alternate heading for a field on a report or graph
IDD FILE	Identify a non-CA-IDMS file in the IDD and build the file and field definition
IDD NAME	Establish the dictionary entity retrieval environment
IDD RECORD	Identify and define CA-IDMS and non-CA-IDMS records
IDD SUBSCHEMA	Identify the subschema and build the file, record, logical record, element record, and field definitions
IDD VERSION	Set a global override of the Options Table VERFILE, VERREC, and VERSCHM defaults
IDMS ACCEPT DBKEY	Transfer database keys to program storage
IDMS ACCEPT PROCEDURE	Return information from the Application Program Information Block (APIB) associated with a database procedure to the program
IDMS ACCEPT STATISTICS	Retrieve the system statistics
IDMS BIND	Sign on the activity with the database management system
IDMS BIND FILE	Give the database management system access to the record in program storage
IDMS BIND PROCEDURE	Establish communications between a program and a DBA-written database procedure
IDMS COMMIT	Request the creation of a checkpoint

Statement	**Usage**
IDMS CONNECT	Establish a record as a member of a set occurrence
IDMS DISCONNECT	Cancel the relationship between a record and a set occurrence
IDMS ERASE	Make a record or logical record unavailable for further processing and remove it from all set occurrences in which it participates as a member
IDMS FIND	Locate a record
IDMS FINISH	Sign off the database management system
IDMS GET	Retrieve current data records
IDMS IF	Test the status of a set
IDMS KEEP	Place a shared or exclusive lock on a record
IDMS MODIFY	Update a record or logical record within the database
IDMS OBTAIN	Locate and then retrieve a record. For database records, see IDMS FIND/OBTAIN. For logical records, see IDMS OBTAIN.
IDMS READY	Establish area availability with the database manager
IDMS RETURN	Retrieve the database key for an indexed record without retrieving the record
IDMS ROLLBACK	Request recovery
IDMS STORE	Place a new record or logical record occurrence into a database
IF	Control the execution of associated statements by testing conditional expressions
INITIATION	A SCREEN procedure invoked during the start of a SCREEN activity
INSERT	Insert a row into a CA-Easytrieve SQL file
JOB	Define and initiate processing activities
JOB INPUT	Identify automatic input to the activity
JOB INPUT NULL	Inhibit automatic input
JOB INPUT SQL	Allow CA-Easytrieve to automatically manage the SQL cursor without a file
KEY	Define valid terminal keys for a screen, specify descriptive text, and assign functions to terminal keys
LINE	Define the content of a report line
LINK	Transfer control from current program to another named program and return to current program
LIST	Regulate the printing or suppression of all statements in the printed

Statement	Usage
	output of a program
LOGICAL-RECORD	Identify the logical records available for automatic or controlled processing of CA-IDMS databases
MACRO	Define the parameters of a macro
MEND	Terminate a macro
MESSAGE	Define message type and text for messages in a SCREEN activity
MOVE	Transfer character strings from one storage location to another
MOVE LIKE	Move contents of fields with identical names from one file or record to another
MSTART	Begin an instream macro
NEWPAGE	Eject the printer to the top of the next page before printing the next line of source program on a statement listing
PARM	Override selected general standards for a program that are set in the Site Options Table
PERFORM	Transfer control to a procedure and return control to next executable statement in current program
POINT	Establish a position within an INDEXED or RELATIVE file from which subsequent data is sequentially retrieved
POP	Restore the previous listing control indicators
PRINT	Produce report output
PROC	Initiate a CA-Easytrieve procedure
PROGRAM	Identify and initiate a processing activity that can optionally initiate JOB, SORT, and SCREEN activities
PUSH	Save the current listing control indicators
PUT	Perform sequential file output
READ	Provide random access to INDEXED and RELATIVE files
RECORD	Identify the CA-IDMS database records available for automatic or controlled processing
REFRESH	Restore the initial screen image by rebuilding it with the current values of program fields
RELEASE	Manually release the hold on any record in an INDEXED or RELATIVE file
REPEAT	Display arrays on a screen

Statement	Usage
REPORT	Define the type and characteristics of a report
REPORT-INPUT	A REPORT procedure that selects or modifies report input data
RESHOW	Re-display a screen image without rebuilding the screen using the current values of program fields
RETRIEVE	Identify the CA-IDMS or IMS/DLI database records that are input to the JOB activity
ROLLBACK	Roll back all recoverable work since the last commit-point
ROW	Specify items to be displayed and received on a row of a screen
SCREEN	Define and initiate a SCREEN activity
SEARCH	Provide access to table data
SELECT (File-based SQL)	Cause a cursor to be declared and opened for a CA-Easytrieve file
SELECT (CA-IDMS)	Specify automatic input of logical records from CA-IDMS databases
SELECT (Non-file SQL)	Identify the rows and columns to be input to a JOB activity when a CA-Easytrieve file is not used
SELECT (Report Selection)	Select report input data
SELECT (Sort Selection)	Select sort input data
SEQUENCE	Specify the order of a report or graph based on the content of one or more fields
SET	Dynamically change screen attributes and control the display of screen errors
SKIP	Space the printer a designated number of lines before printing the next line of a statement listing
SORT	Sequence an input file in alphabetical or numerical order based on fields specified as keys
SQL	Indicate a valid SQL statement for any of the supported SQL database management systems
SQL INCLUDE	Indicate SQL table information is used to generate CA-Easytrieve field definitions
STOP	Terminate activities
SUM	Specify the quantitative fields which are totaled for a control report

Statement	Usage
TERMINATION (Reports)	A REPORT procedure invoked at the end of a report, commonly used to print report footing information
TERMINATION (Screens)	A SCREEN procedure invoked once during the end of a SCREEN activity, used to perform actions that are executed once at the end of the activity
TITLE (Graphs)	Specify a title to be displayed on a graph
TITLE (Reports)	Define an optional report title and its position on a title line
TITLE (Screens)	Define and center title items on a screen
TRANSFER	Transfer execution to a program without returning to the invoking program
UPDATE	Update a row from a CA-Easytrieve SQL file
VALUE	Specify fields used to draw a graph
WRITE	Update and delete existing records or add new records to INDEXED and RELATIVE files

Functional Category Summary

CA-Easytrieve statements are divided into the following functional categories:

- Library Definition
- File Management
- Screen Processing
- Report Processing
- Graph Processing
- Generalized Programming
- Inter-program Execution
- Decision and Branching Logic
- Listing Control
- Assignment and Moves
- Macro Processing
- Native SQL
- CA-IDMS Database Processing
- IMS/DLI Database Processing.

Under each category is a list of the statements that belong to each. Also given is a brief overview of the usage of each statement.

Library Definition

Statement	Usage
COPY	Duplicate field definitions of a named file
DECLARE	Name a set of screen attributes or an input edit pattern, or whether a program is statically or dynamically linked
DEFINE	Specify a data field within a file or within working storage
FILE	Describe a file and database references
SQL INCLUDE	Indicate SQL table information is used to generate CA-Easytrieve field definitions

File Management

Statement	Usage
CLOSE	Close a file
COMMIT	Commit a logical unit of recoverable work
DELETE	Delete a row from a CA-Easytrieve SQL file
DISPLAY	Format and transfer data to the system output device or to a named file
ENDTABLE	Delimit instream data used to create small tables
FETCH	Retrieve a row from a CA-Easytrieve SQL file
GET	Place the next sequential record of the named file into the file's record buffer
INSERT	Insert a row into a CA-Easytrieve SQL file
JOB	Define and initiate processing activities
JOB INPUT NULL	Inhibit automatic input
JOB INPUT SQL	Allow CA-Easytrieve to automatically manage the SQL cursor without a file
POINT	Establish a position within an INDEXED or RELATIVE file from which subsequent data is sequentially received
PUT	Perform sequential file output

Statement	Usage
READ	Provide random access to INDEXED and RELATIVE files
RELEASE	Manually release the hold on any record in an INDEXED or RELATIVE file
ROLLBACK	Roll back all recoverable work since the last commit-point
SELECT (File-based SQL)	Cause a cursor to be declared and opened for a CA-Easytrieve file
SELECT (Non-file SQL)	Identify the rows and columns to be input to a JOB activity when a CA-Easytrieve file is not used
SELECT (Sort Selection)	Select sort input data
SORT	Sequence an input file in alphabetical or numerical order based on fields specified as keys
UPDATE	Update a row from a CA-Easytrieve SQL file
WRITE	Update and delete existing records or add new records to INDEXED and RELATIVE files

Screen Processing

Statement	Usage
AFTER-SCREEN	A SCREEN procedure performed after a SCREEN activity receives data from the terminal
BEFORE-SCREEN	A SCREEN procedure invoked before a SCREEN activity sends data to the terminal
CURSOR	Set the initial position of the screen cursor
DEFAULT	Override system-defined screen attributes and message locations
EXIT	Terminate a SCREEN activity
INITIATION	A SCREEN procedure invoked once during the start of a SCREEN activity
KEY	Define valid terminal keys for a screen, specify descriptive text, and assign functions to terminal keys
MESSAGE	Define message type and text for messages in a SCREEN activity
REFRESH	Restore the initial screen image by rebuilding it with the current values of program fields
REPEAT	Display arrays on a screen

Statement	Usage
RESHOW	Re-display a screen image without rebuilding the screen using the current values of program fields
ROW	Specify items to be displayed and received on a row of a screen
SCREEN	Define and initiate a SCREEN activity
SET	Dynamically change screen attributes and control the display of screen errors
TERMINATION	A SCREEN procedure invoked once during the end of a SCREEN activity, used to perform actions that are executed once at the end of the activity
TITLE	Define and center title items on a screen

Report Processing

Statement	Usage
AFTER-BREAK	A REPORT procedure invoked following the printing of summary lines for a control break
AFTER-LINE	A REPORT procedure invoked after printing a detail line on a report
BEFORE-BREAK	A REPORT procedure invoked before printing the summary lines for a control break
BEFORE-LINE	A REPORT procedure invoked before printing a detail line on a report
CONTROL	Identify control fields used in a control report
ENDPAGE	A REPORT procedure used to produce page footing information
HEADING	Define an alternate heading for a field on a report
LINE	Define the content of a report line
PRINT	Produce report output
REPORT	Define the type and characteristics of a report
REPORT-INPUT	A REPORT procedure that selects and/or modifies report input data
SELECT	Select report input data
SEQUENCE	Specify the order of a report based on the content of one or more fields
SUM	Specify the quantitative fields which are totaled for a control report
TERMINATION	A REPORT procedure invoked at the end of a report, commonly used to print report footing information

Statement	Usage
TITLE	Define an optional report title and its position on a title line

Graph Processing

Statement	Usage
DRAW	Produce graphic output by initiating a GRAPH subactivity
GRAPH	Define the style and characteristics of a graph
HEADING	Define an alternate heading for a field on a graph
SEQUENCE	Specify the order of a graph based on the content of one or more fields
TITLE	Specify a title to be displayed on a graph
VALUE	Specify fields used to draw a graph

Generalized Programming

Statement	Usage
PARM	Override selected general standards for a program that are set in the Site Options Table
PROGRAM	Identify and initiate a processing activity that can optionally initiate JOB, SORT, and SCREEN activities

Inter-program Execution

Statement	Usage
CALL	Invoke subprograms written in other programming languages
LINK	Transfer control from current program to another named program and return to current program
TRANSFER	Transfer execution to a program without returning to the invoking program

Decision and Branching Logic

Statement	Usage
CASE	Conditionally execute one of several alternative groups of statements based on the value of a specific field
DO UNTIL	Control repetitive program logic by evaluating the condition at the bottom of a group of statements
DO WHILE	Control repetitive program logic by evaluating the condition at the top of a group of statements
ELSE	Identify statements to be executed when IF conditions are false. See IF.
ELSE-IF	Identify a conditional expression to be tested when the previous IF or ELSE-IF conditional expression is false. See IF.
END-DO	Terminate the body of a loop associated with a DO UNTIL or DO WHILE statement. See DO UNTIL and DO WHILE.
END-IF	Terminate the logic associated with the previous IF statement. See IF.
EXECUTE	Invoke a JOB, SORT, or SCREEN activity from a PROGRAM or SCREEN activity
GOTO	Modify the top to bottom logic flow of statement execution
GOTO JOB	Branch to the top of the current JOB activity
GOTO SCREEN	Branch to the top of the current SCREEN activity
IF	Control the execution of associated statements by testing conditional expressions
PERFORM	Transfer control to a procedure and return control to next executable statement in current program
STOP	Terminate activities

Listing Control

Statement	Usage
*	Document comments in a program
LIST	Regulate the printing or suppression of all statements in the printed output of a program
NEWPAGE	Eject the printer to the top of the next page before printing the next line of source program on a statement listing
POP	Restore the previous listing control indicators
PUSH	Save the current listing control indicators

Statement	Usage
SKIP	Space the printer a designated number of lines before printing the next line of a statement listing

Assignment and Moves

Statement	Usage
Assignment	Establish a value in a field
MOVE	Transfer character strings from one storage location to another
MOVE LIKE	Move contents of fields with identical names from one file to another

Macro Processing

Statement	Usage
%	Invoke a macro
ACCESS	Access a macro secured against unauthorized access in CA-PANVALET or VSAM
MACRO	Define the parameters of a macro
MEND	Terminate a macro
MSTART	Begin an instream macro

Native SQL

Statement	Usage
SQL	Indicate a valid SQL statement for any of the supported SQL database management systems

CA-IDMS Database Processing

Statement	Usage
ELEMENT-RECORD	Identify the element records that comprise the logical record
IDD FILE	Identify a non-CA-IDMS file in the IDD and build the file and field

Statement	Usage
	definition
IDD NAME	Establish the dictionary entity retrieval environment
IDD RECORD	Identify and define CA-IDMS and non-CA-IDMS records
IDD SUBSCHEMA	Identify the subschema and build the file, record, logical record, element record, and field definitions
IDD VERSION	Set a global override of the Options Table VERFILE, VERREC, and VERSCHM defaults
IDMS ACCEPT DBKEY	Transfer database keys to program storage
IDMS ACCEPT PROCEDURE	Return information from the Application Program Information Block (APIB) associated with a database procedure to the program
IDMS ACCEPT STATISTICS	Retrieve the system statistics
IDMS BIND	Sign on the activity with the database management system
IDMS BIND FILE	Give the database management system access to the record in program storage
IDMS BIND PROCEDURE	Establish communications between a program and a DBA-written database procedure
IDMS COMMIT	Request the creation of a checkpoint
IDMS CONNECT	Establish a record as a member of a set occurrence
IDMS DISCONNECT	Cancel the relationship between a record and a set occurrence
IDMS ERASE	Make a record or logical record unavailable for further processing and remove it from all set occurrences in which it participates as a member
IDMS FIND	Locate a record
IDMS FINISH	Sign off the database management system
IDMS GET	Retrieve current data records
IDMS IF	Test the status of a set
IDMS KEEP	Place a shared or exclusive lock on a record
IDMS MODIFY	Update a record or logical record within the database
IDMS OBTAIN	Locate and then retrieve a record. For database records, see IDMS FIND/OBTAIN. For logical records, see IDMS OBTAIN.
IDMS READY	Establish area availability with the database manager
IDMS RETURN	Retrieve the database key for an indexed record without retrieving the

Statement	Usage
	record
IDMS ROLLBACK	Request recovery
IDMS STORE	Place a new record or logical record occurrence into the database
LOGICAL-RECORD	Identify the logical records available for automatic or controlled processing of CA-IDMS databases
RECORD	Identify the CA-IDMS database records available for automatic or controlled processing
RETRIEVE	Identify the CA-IDMS database records that are input to the JOB activity
SELECT (CA-IDMS)	Specify automatic input of logical records from CA-IDMS databases

IMS/DLI Database Processing

Statement	Usage
DLI	Perform IMS/DL/I functions against an IMS/DL/I database

Chapter

3

Statements A - C

% (Macro Invocation) Statement

The macro invocation statement consists of a macro name preceded by a percent (%) sign. Its syntax is:

Syntax

```
%macro-name  [positional-parameters]...[keyword-parameters]
```

Parameters

`%macro-name` Macro-name is the name of a previously stored macro that you want to invoke.

`[positional-parameters]` Supply values of positional parameters in the macro. You must supply positional parameters before any keyword parameters.

`[keyword-parameters]` Supply both the keyword and values of keyword parameters in the macro.

Usage Notes

See the MACRO Statement for more information on defining substitutable parameters in the macro. See the *CA-Easytrieve Programmer Guide* for complete information on using macros.

* (Comment) Statement

You can document your CA-Easytrieve programs with a comment statement. When the first non-blank character of a statement is an asterisk (*), the remainder of that record is a comment statement.

Syntax

```
* comment-text
```

Usage Notes

You can use comment statements any place within a program, except within a continued statement. A statement containing all blanks is treated as a comment.

If you code comment statements within a SCREEN declaration and maintain the screen with the CA-Easytrieve/Online Screen Painter, all comment statements are moved to the top of the declaration.

ACCESS Statement

The ACCESS statement enables you to access a macro secured against unauthorized access in CA-Panvalet or VSAM.

Syntax

CA-Panvalet

```
ACCESS 'eight-byte code'
```

VSAM

```
ACCESS 'eight-byte password'
```

'*eight-byte code*' A security access code applies to an individual CA-Panvalet library member. You must supply the security access code on an ACCESS record before CA-Easytrieve can retrieve a secured member.

'*eight-byte password*' VSAM provides the capability of protecting the macro library through the use of VSAM password protection. Before CA-Easytrieve can retrieve a macro from a secured library, you must supply the library password on an ACCESS record prior to the first macro call.

Usage Notes

For both CA-Panvalet and VSAM macro storage access methods, the ACCESS record can appear anywhere in the CA-Easytrieve program prior to the retrieval of the macro, and remains in effect until the next ACCESS record is encountered. The ACCESS record must be on a record by itself. CA-Easytrieve does not print the ACCESS record.

AFTER-BREAK Report Procedure

An AFTER-BREAK procedure is invoked following the printing of summary lines for a control break. It can be used to produce special annotation on control reports.

Syntax

```
AFTER-BREAK. PROC
```

Usage Notes

The AFTER-BREAK procedure is invoked once for each level of break. For example, assume two control fields are specified. When the minor field causes a control break, the AFTER-BREAK procedure is invoked only once. When the major field causes a control break, AFTER-BREAK is invoked twice.

The value of LEVEL (a system-defined field) can be used to determine which control break is being processed. The value of BREAK-LEVEL (a system-defined field) contains the number of the field causing the control break. TALLY (a system-defined field) contains the number of records in a particular control group. See the CONTROL Statement for more information. See the *CA-Easytrieve Programmer Guide* for examples of LEVEL and BREAK-LEVEL.

Note: If NOPRINT is specified on a CONTROL statement, the AFTER-BREAK procedure is still executed.

An AFTER-BREAK procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

In the following example, the total line for the control field STATE receives special annotation.

```
Statements:

 FILE FILE1
 LAST-NAME  1  5 A
 STATE      6  2 A
 ZIP        8  5 N
 PAY-NET    13 5 N 2
 JOB INPUT FILE1 NAME MYPROG
   PRINT REPORT1
 *
 REPORT REPORT1 LINESIZE 65 +
   SUMMARY  SUMCTL DTLCOPY
   SEQUENCE STATE ZIP LAST-NAME
   CONTROL  STATE ZIP
   LINE 01  LAST-NAME STATE ZIP PAY-NET
 *
 AFTER-BREAK. PROC
```

```
   IF LEVEL EQ 2
     DISPLAY 'TOTALS FOR THE STATE OF ' STATE
   END-IF
 END-PROC

 Data:

BROWNIL6007612345
 BROWNIL6007667890
 JONESIL6007709876
 JONESIL6007754321
 SMITHTX7521811111
 SMITHTX7521866666

Results:

              LAST-NAME   STATE    ZIP      PAY-NET

                BROWN      IL     60076       802.35
                JONES      IL     60077       641.97
                           IL                1444.32
 TOTALS FOR THE STATE OF IL

                SMITH      TX     75218       777.77
                           TX                 777.77
 TOTALS FOR THE STATE OF TX
                                             2222.09
```

AFTER-LINE Report Procedure

An AFTER-LINE procedure is invoked immediately following the printing of each detail line on a report. An AFTER-LINE procedure is commonly used to print a literal string after a detail line on the report.

Syntax

```
AFTER-LINE. PROC
```

Usage Notes

The AFTER-LINE procedure is invoked after each individual line in a line group. The system-defined field LINE-NUMBER contains the number of the line in the group being processed.

Note that when using an AFTER-LINE procedure, the detail line for the report has already been built. You cannot modify the contents of the detail line with an AFTER-LINE procedure. (To modify the contents of a detail line on a report, use a REPORT-INPUT or BEFORE-LINE procedure.)

An AFTER-LINE procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

The following example illustrates how an AFTER-LINE procedure can cause information to be printed following a detail line of a report:

```
Statements:

 FILE FILE1
 LAST-NAME  1  5 A
 STATE      6  2 A
 ZIP        8  5 N
 PAY-NET    13 5 N 2
 JOB INPUT FILE1 NAME MYPROG
   PRINT REPORT1
 *
 REPORT REPORT1 LINESIZE 65 +
   DTLCTL   EVERY
   SEQUENCE STATE ZIP LAST-NAME
   CONTROL  STATE ZIP
   LINE 01  LAST-NAME STATE ZIP PAY-NET
 *
 AFTER-LINE. PROC
   IF PAY-NET GE 500
     DISPLAY '* EMPLOYEE ' LAST-NAME ' +
      EXCEEDED WEEKLY SALARY GOAL *'
   END-IF
 END-PROC

 Data:

 BROWNIL6007612345
 BROWNIL6007667890
 JONESIL6007709876
 JONESIL6007754321
 SMITHTX7521811111
 SMITHTX7521866666

Results:

            LAST-NAME   STATE    ZIP     PAY-NET

              BROWN      IL     60076      678.90
* EMPLOYEE BROWN EXCEEDED WEEKLY SALARY GOAL *
              BROWN      IL     60076      123.45
                         IL     60076      802.35

              JONES      IL     60077      543.21
* EMPLOYEE JONES EXCEEDED WEEKLY SALARY GOAL *
              JONES      IL     60077       98.76
                         IL     60077      641.97

                         IL               1444.32

              SMITH      TX     75218      666.66
* EMPLOYEE SMITH EXCEEDED WEEKLY SALARY GOAL *
              SMITH      TX     75218      111.11
                         TX     75218      777.77

                         TX                777.77

                                          2222.09
```

AFTER-SCREEN Screen Procedure

An AFTER-SCREEN procedure is invoked after the screen activity receives data from the terminal.

Syntax

```
AFTER-SCREEN. PROC
```

Usage Notes

All branch actions (REFRESH, RESHOW, EXIT, GOTO SCREEN) are valid in the AFTER-SCREEN procedure and any procedure performed by the AFTER-SCREEN procedure. The AFTER-SCREEN procedure is not executed if the key pressed is assigned to execute a branch action with a KEY statement. You typically use an AFTER-SCREEN procedure to perform complex editing and to perform I/O after data entry.

An AFTER-SCREEN procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

```
SCREEN NAME SCRN1
    KEY F3 NAME 'Exit' EXIT
    KEY F8 NAME 'Forward'
. . .
    AFTER-SCREEN. PROC
      GET PERSNL
      IF EOF PERSNL
        EXIT
      END-IF
    END-PROC
```

Assignment Statement

The Assignment statement establishes a value in a field. The value can be a copy of the data in another field or literal, or it can be the result of an arithmetic or logical expression evaluation.

The two formats of the Assignment statement are:

Syntax

Format 1 (Normal Assignment)

```
                             [ROUNDED  ]{= } {send-field-name      }
     receive-field-name [INTEGER] [          ]{  } {send-literal         }
                             [TRUNCATED]{EQ} {arithmetic-expression}
```

Format 2 (Logical Expression)

```
                        {= }                  {AND} {bit-mask-field-name}
     receive-field-name {  } send-field-name {OR } {                   }
                        {EQ}                  {XOR} {bit-mask-literal   }
```

Parameters

Format 1 (Normal Assignment)

receive-field-name Specify the field name to which a value will be assigned.

[INTEGER] Specify INTEGER to ignore the fractional portion of the value being assigned. INTEGER causes only the numerics to the left of the decimal point to be transferred during the assignment.

```
[ROUNDED  ]
[TRUNCATED]
```

Specify ROUNDED or TRUNCATED when the receiving field (*receive-field-name*) is too small to handle the fractional result of the assignment. TRUNCATED is the default.

Specify ROUNDED to round off the fractional result of the assignment statement. The least significant digit of the result (receiving field) has its value increased by one when the most significant digit of the excess decimal digits is greater than or equal to five. For example, if 10.75 is the value of the sending field and the receiving field has one decimal place, ROUNDED causes the receiving field to be 10.8.

Specify TRUNCATED to truncate the result of the assignment statement. Low order digits are truncated on the right as necessary when the result is moved to the receiving field.

If INTEGER is used with ROUNDED, the result is rounded to the nearest integer before the INTEGER function is performed. If INTEGER is used with TRUNCATED (the default), then only the INTEGER function is performed.

Note: INTEGER, ROUNDED, and TRUNCATED are valid only with numeric fields.

```
{= }
{  }
{EQ}
```

Use EQ or = to indicate equivalency.

```
{send-field-name      }
{send-literal         }
```

```
{arithmetic-expression}
```

Send-field-name names the field that is copied to *receive-field-name*.

Send-literal contains the literal that is copied to *receive-field-name*.

Arithmetic-expression contains numeric values separated by arithmetic operators (+, -, *, /). The result of the *arithmetic-expression* is placed in *receive-field-name*.

Format 2 (Logical Expression)

receive-field-name Specify the field name to which a value will be assigned.

```
{= }
{  }
{EQ}
```

Use EQ or = to indicate equivalency.

send-field-name *Send-field-name* names the field that is copied to *receive-field-name*.

```
{AND}
{OR }
{XOR}
```

Specify AND, OR, or XOR.

- AND - Zero bits in *bit-mask-field-name* or *bit-mask-literal* are carried forward to *send-field-name* and the result is placed in *receive-field-name*.
- OR - One bits in *bit-mask-field-name* or *bit-mask-literal* are carried forward to *send-field-name* and the result is placed in *receive-field-name*.
- XOR - Corresponding bits of *bit-mask-field-name* or *bit-mask-literal*, and *send-field-name* must be opposite (zero and one) to result in a one bit in *receive-field-name*.

```
{bit-mask-field-name}
{                   }
{bit-mask-literal   }
```

Bit-mask-field-name is the name of a field that is logically combined with *send-field-name*, the result of which is carried forward to *receive-field-name*.

Bit-mask-literal is a literal bit mask that is logically combined with *send-field-name*, the result of which is carried forward to *receive-field-name*.

Usage Notes

Format 1 (Normal Assignment)

Format 1 sets the value of *receive-field-name* equal to the value of *send-field-name*, *send-literal*, or the arithmetic expression. See Assignments and Moves in the "Coding a CA-Easytrieve Program" chapter of the *CA-Easytrieve Programmer Guide* for complete rules of the Assignment statement.

Note: See the *CA-Easytrieve Programmer Guide* for complete Assignment statement rules for converting from EBCDIC to DBCS.

Format 2 (Logical Expression)

Format 2 of the Assignment statement sets the value of *receive-field-name* equal to the result of evaluating a logical expression. The value of *send-field-name* is logically acted upon by the value of *bit-mask-field-name* or *bit-mask-literal*. The lengths of all values must be the same and *bit-mask-literal* must be hexadecimal.

Note: If *receive-field-name* is nullable, then its indicator is set to zero, indicating NOT NULL. If any operands on the right-hand side contain NULLs, a runtime error occurs.

Examples

The following examples of the Assignment statement illustrate its various rules.

The first example shows assignment format 1 when the receive-field-name is alphanumeric:

```
Format 1 (Normal Assignment)

DEFINE F1A   W   4   A
DEFINE F2A1  W   1   A    VALUE 'A'
DEFINE F2A2  W   6   A    VALUE 'ABCDEF'
DEFINE F2N1  W   2   N    VALUE 12
DEFINE F2N2  W   3   P 1  VALUE 1234.5
 ...
                    Resulting Value

F1A  =  F2A1             'A   '
F1A  =  F2A2             'ABCD'
F1A  =  F2N1             '0012'
F1A  =  F2N2             '2345'
F1A  =  X'FF'            X'FF404040'
```

Note: For an example using varying length alphanumeric fields, see Field Definition in the *CA-Easytrieve Programmer Guide*.

This example shows assignment format 1 when the receive-field-name is numeric:

```
Format 1 (Normal Assignment)

Statements:

DEFINE F1N  W 4 N 1
DEFINE F2N1 W 4 N 1 VALUE 1
DEFINE F2N2 W 4 N 1 VALUE 2
DEFINE F2N3 W 4 N 1 VALUE 3
JOB INPUT NULL NAME MYPROG
   F1N = F2N1 + F2N2 + F2N3
   DISPLAY SKIP 2 +
          'F1N = F2N1 + F2N2 + F2N3         = ' F1N
   F1N = F2N1 + F2N2 / F2N3
   DISPLAY SKIP 2 +
          'F1N = F2N1 + F2N2 / F2N3         = ' F1N
   F1N = (F2N1 + F2N2) / F2N3
   DISPLAY SKIP 2 +
          'F1N = (F2N1 + F2N2) / F2N3       = ' F1N
   F1N = ((F2N1 / F2N2) * 100) + .5
   DISPLAY SKIP 2 +
          'F1N = ((F2N1 / F2N2) * 100) + .5 = ' F1N
STOP

Results:
                                           Resulting
                                             Value

         F1N = F2N1 + F2N2 + F2N3         =   6.0
               (1   +   2  +  3)

         F1N = F2N1 + F2N2 / F2N3         =   1.6
               (1   + 2    / 3)
               (1   + 0.6666)

         F1N = (F2N1 + F2N2) / F2N3       =   1.0
               (( 1  + 2)    / 3)
                     (3      / 3)

         F1N = ((F2N1 / F2N2) * 100) + .5 =  50.5
               ((   1 / 2)    * 100) + .5
                    ((0.5     * 100) + .5)
                                (50  + .5)
```

The following example illustrates the use of the INTEGER, ROUNDED, and TRUNCATED parameters.

If:

```
SENDFLD  W  5  N  2  VALUE(10.75)
RCVFLD   W  5  N  1
```

Then:

Assignment Statement	**RCVFLD Result**
RCVFLD INTEGER ROUNDED = SENDFLD	11.0
RCVFLD INTEGER TRUNCATED = SENDFLD	10.0
RCVFLD INTEGER = SENDFLD	10.0
RCVFLD ROUNDED = SENDFLD	10.8
RCVFLD TRUNCATED = SENDFLD	10.7
RCVFLD = SENDFLD	10.7

```
Format 2 (Logical Expression Evaluation)

Statements:

DEFINE F1P  W 2 P  MASK HEX
DEFINE F2P  W 2 P  VALUE X'123D'
JOB INPUT NULL NAME MYPROG
   F1P = F2P AND X'FFFE'
   DISPLAY SKIP 2  +
          'F1P = F2P AND X''FFFE'' = ' F1P
   F1P = F2P OR  X'000F'
   DISPLAY SKIP 2 +
          'F1P = F2P OR  X''000F'' = ' F1P
   F1P = F2P XOR X'FFFF'
   DISPLAY SKIP 2 +
          'F1P = F2P XOR X''FFFF'' = ' F1P
   F1P = F2P XOR F2P
   DISPLAY SKIP 2 +
          'F1P = F2P XOR F2P       = ' F1P
STOP

Results:
                           Resulting
                             Value

       F1P = F2P AND X'FFFE' = 123C

       F1P = F2P OR  X'000F' = 123F

       F1P = F2P XOR X'FFFF' = EDC2

       F1P = F2P XOR F2P      = 0000
```

ATTR Parameter

The ATTR parameter is used to assign screen attributes to a field or literal. You can specify a DECLAREd screen attribute name or a list of attribute keywords. The ATTR parameter can be used in the following statements:

- DECLARE
- DEFAULT
- ROW
- TITLE

When used in these statements the ATTR parameter completely overrides any site or screen default attributes. See the DEFAULT Statement to set default screen attributes that override site attributes

Syntax

```
ATTR  [attribute-name  ]
      [(attribute-list)]
```

```
attribute-list

[SENDONLY] +

[CURSOR] +

[ASKIP  ] +
[PROTECT]

[NUMERIC] +

[INTENSE  ] +
[INVISIBLE]

[GREEN         ]
[RED           ]
[BLUE          ]
[TURQ|TURQUOISE] +
[PINK          ]
[YELLOW        ]
[BLACK         ]
[WHITE         ]

[MUSTFILL] +

[MUSTENTER] +

[TRIGGER] +

[BLINK    ]
[REVERSE  ] +
[UNDERLINE]

[ALARM] +

[BOX  ]
[LEFT ]
[RIGHT]
[UNDER]
[OVER ]
```

Parameters

attribute-name Specify a DECLAREd screen attribute name. See the DECLARE Statement for more information.

[SENDONLY] The SENDONLY parameter specifies that the field is not to be received. The field is ignored if entered. SENDONLY is implied for literals.

[CURSOR] Specify CURSOR to place the cursor on this field when displayed on the terminal. If more than one field contains the CURSOR attribute, the cursor is placed on the first field that contains CURSOR.

CURSOR is ignored for literals.

Note: The cursor cannot be moved into a field that also contains the ASKIP, PROTECT, or SENDONLY attributes.

```
[ASKIP  ]
```

```
[PROTECT]
```

ASKIP specifies that the field is an auto-skip field. PROTECT specifies that the field is protected and not auto-skipped. If neither is specified, the field is unprotected.

ASKIP is implied for literals.

[NUMERIC] NUMERIC specifies that only numeric data can be entered in this screen field. Use NUMERIC for permitting only numeric data in alphanumeric fields. NUMERIC is implied for all numeric data types, and ignored for literals.

```
[INTENSE  ]
[INVISIBLE]
```

INTENSE specifies that the field displays brightly. INVISIBLE specifies that the field is present on the screen but is not displayed. INVISIBLE is ignored for literals.

On 3270 extended attribute terminals, INTENSE is ignored when a color attribute is also specified.

```
[GREEN         ]
[RED           ]
[BLUE          ]
[TURQ|TURQUOISE]
[PINK          ]
[YELLOW        ]
[BLACK         ]
[WHITE         ]
```

The value specified is the color of the field or literal when displayed on a screen. If no color is specified, hardware defaults apply.

Note: BLACK is valid on the workstation only. It is ignored on the mainframe. The color BLACK is flagged as an error if the portability switch (/P) is specified for the workstation compiler.

[MUSTFILL] Specify MUSTFILL to require that all spaces have a non-blank character typed into them. MUSTFILL is ignored for literals and on terminals that do not support a mandatory-fill attribute.

[MUSTENTER] Use MUSTENTER to send an error message to the terminal if the field was not changed. MUSTENTER is ignored for literals and on terminals that do not support a mandatory-enter attribute. MUSTENTER is ignored for literals.

[TRIGGER] TRIGGER causes the screen to be received as soon as the terminal operator has modified the field and tries to move the cursor out of the field. TRIGGER is ignored for literals and on terminals that do not support a trigger attribute.

```
[BLINK    ]
[REVERSE  ]
[UNDERLINE]
```

BLINK displays the item blinking. REVERSE displays the item in reverse video. UNDERLINE displays the item underlined.

`[ALARM]` ALARM causes the terminal alarm to sound.

ALARM is ignored for literals.

```
[BOX  ]
[LEFT ]
[RIGHT]
[UNDER]
[OVER ]
```

BOX specifies that field outlining displays a box surrounding the field.

LEFT specifies that field outlining displays a vertical line to the left of a field.

RIGHT specifies that field outlining displays a vertical line to the right of a field.

UNDER specifies that field outlining displays a horizontal line below a field.

OVER specifies that field outlining displays a horizontal line above a field.

Note: BOX, LEFT, RIGHT, UNDER, and OVER are ignored on terminals that do not support outlining attributes.

Usage Notes

The ATTR parameter can be specified without an *attribute-name* or an *attribute-list* only on the DECLARE statement. This permits a named attribute to be declared and then assigned later in the program. A runtime error occurs if a named attribute is used without any attributes assigned to it. The DEFAULT, ROW, and TITLE statements require an *attribute-name* or an *attribute-list* after the ATTR keyword.

BEFORE-BREAK Report Procedure

A BEFORE-BREAK procedure is invoked before printing the summary lines for a control break. It can be used to calculate percentages and average totals. These values must be calculated immediately before printing.

Syntax

```
BEFORE-BREAK. PROC
```

Usage Notes

The BEFORE-BREAK procedure is invoked once for each level of break. For example, assume two control fields are specified. When the minor field causes a control break, the BEFORE-BREAK procedure is invoked only once. When the major field causes a control break, BEFORE-BREAK is invoked twice.

The value of LEVEL (a system-defined field) can be used to determine which control break is being processed. The value of BREAK-LEVEL (a system-defined field) contains the number of the field causing the control break. TALLY (a system-defined field) contains the number of records in a particular control group. See the CONTROL Statement for more information. See the *CA-Easytrieve Programmer Guide* for examples of LEVEL and BREAK-LEVEL.

Note: If NOPRINT is specified on a CONTROL statement, the BEFORE-BREAK procedure is still executed.

A BEFORE-BREAK procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

Consider the following percentage calculation, paying special attention to when and how PERCENT is calculated:

```
FILE FILE1 FB(80 8000)
LAST-NAME  1  5 A
STATE      6  2 A
ZIP        8  5 N
PAY-NET   13  5 N 2
*
PERCENT    W  2 N 2
TOTAL-NET  S  8 N 2
*
JOB INPUT FILE1 NAME MYPROG
*
  TOTAL-NET = TOTAL-NET + PAY-NET
  PRINT REPORT1
*
REPORT REPORT1 LINESIZE 80 +
  SUMMARY  SUMCTL DTLCOPY
  SEQUENCE STATE ZIP LAST-NAME
  CONTROL  STATE ZIP
  LINE 01 LAST-NAME STATE ZIP PAY-NET PERCENT
*
BEFORE-BREAK. PROC
  PERCENT = PAY-NET * 100 / TOTAL-NET
END-PROC

Data:

BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
```

```
SMITHTX7521866666

Results:

LAST-NAME   STATE    ZIP     PAY-NET     PERCENT

  BROWN      IL     60076     802.35     36.10
  JONES      IL     60077     641.97     28.89
             IL              1444.32     64.99

  SMITH      TX     75218     777.77     35.00
             TX               777.77     35.00

                             2222.09    100.00
```

The BEFORE-BREAK procedure computes the percentage for each control break by multiplying the sum of PAY-NET by 100 and then dividing by TOTAL-NET.

Note: TOTAL-NET is a static (S) working storage field summed in the JOB activity processing.

BEFORE-LINE Report Procedure

A BEFORE-LINE procedure is invoked immediately before the printing of each detail line on a report. A BEFORE-LINE procedure is commonly used to print a literal string before a detail line on the report or to change the contents of the detail line before printing.

Syntax

```
BEFORE-LINE. PROC
```

Usage Notes

The BEFORE-LINE procedure is invoked before each individual line in a line group. The system-defined field LINE-NUMBER contains the number of the line in the group being processed.

A BEFORE-LINE procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

See the AFTER-LINE Report Procedure for an example.

BEFORE-SCREEN Screen Procedure

A BEFORE-SCREEN procedure is invoked before the screen activity sends the data to the terminal. It precedes building the screen and the terminal I/O process.

Syntax

```
BEFORE-SCREEN. PROC
```

Usage Notes

You typically use a BEFORE-SCREEN procedure to perform I/O, initialize screen fields or set the cursor position.

GOTO SCREEN, REFRESH, and RESHOW are invalid in the BEFORE-SCREEN procedure, and in any procedure performed by the BEFORE-SCREEN procedure.

A BEFORE-SCREEN procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

```
SCREEN NAME SCRN1
  KEY F3 NAME 'Exit' EXIT
  KEY F8 NAME 'Forward'
. . .
  BEFORE-SCREEN. PROC
     GET PERSNL
     IF EOF PERSNL
       EXIT
     END-IF
  END-PROC
```

CALL Statement

The CALL statement provides a means to dynamically or statically invoke subprograms written in other programming languages.

Syntax

```
                  [        {field-name}     ]
CALL program-name [USING ( {           } ...)] [RETURNS return-field]
                  [        {'literal'  }     ]
```

Parameters

program-name *Program-name* is the name of the subprogram that you want invoked. It is loaded into storage as part of an activity initiation.

```
[        {field-name}     ]
[USING ( {          } ...)]
[        {'literal' }     ]
```

USING specifies the parameter list passed to the subprogram.

Field-name must identify a system-defined field, a working storage field, or a field defined in an accessible file.

'Literal' can be any alphanumeric literal that is passed to the program.

[RETURNS *return-field*] RETURNS identifies a numeric field that will contain the return code passed back by a called subprogram. If calling a COBOL subprogram, the return code is the value in the COBOL RETURN-CODE field. If calling an Assembler subprogram, the return code is the value contained in register 15 on the mainframe, the AX register on the workstation. If coding a C subprogram, the return code is the value returned from the function.

Return-field is a numeric CA-Easytrieve field which contains the RETURNed value. The field can be a user-defined field or you can use the system-defined field, RETURN-CODE, to pass the return code to the operating system.

Usage Notes

The program being CALLed can either be statically or dynamically bound with your CA-Easytrieve program. The way that the CALLed program is bound is determined by the following, in order:

1. If the program was declared on a DECLARE statement, the STATIC or DYNAMIC keyword on the DECLARE statement determines how it is bound.
2. If specified, the CALL parameter on the PARM statement supplies the default for all CALLed programs in your CA-Easytrieve program.
3. The default is determined by the environment. The default on the mainframe is DYNAMIC. The default on the workstation and UNIX is STATIC.

COBOL programs cannot be called by CA-Easytrieve programs in the CICS environment. See the *CICS Programmer's Reference Manual* for more information.

Any mainframe program being CALLed in a CICS environment must execute in conversational mode. The task must not be terminated by a CALLed program.

See the *CA-Easytrieve Programmer Guide* for complete details of subprogram linkage.

Examples

The first example shows a CALL statement without parms, the second show on with parms:

```
CALL ASMPGM
```

```
CALL ASMPGM USING ('USERFIL', USERFLD)
```

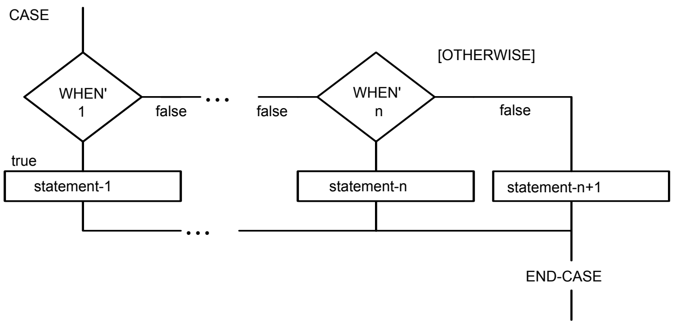
CASE and END-CASE Statements

The CASE and END-CASE statements are used to conditionally execute one of several alternative groups of statements based on the value of a specific field.

Syntax

```
CASE    field-name

  WHEN compare-literal-1 [THRU range-literal-1]  [...]
    statement-1

  WHEN compare-literal-n [THRU range-literal-n]  [...]
    statement-n

  [OTHERWISE      ]
  [  statement-n+1]

END-CASE
```

The following diagram illustrates CASE statement logic:



Parameters

field-name *Field-name* specifies a field that contains a value that is compared to the values represented by *compare-literal* [*THRU range-literal*].

Field-name can be a field of any type. If *field-name* is numeric, it must have zero or no decimal places.

WHEN You can specify as many WHEN conditions as necessary. At least one WHEN condition is required. You cannot code statements between CASE and the first WHEN condition. You must supply a unique set of values to be compared with *field-name* in each WHEN condition.

***compare-literal* [THRU *range-literal*]** *Compare-literal* is the value to be compared with *field-name*. You can specify a single literal, a series of literals, or a range of literals. A range is represented by *compare-literal THRU range-literal*. A range is satisfied when *field-name* is greater than or equal to the lesser of *compare-literal* and *range-literal* and is less than or equal to the greater of *compare-literal* and *range-literal*.

When *field-name* is alphanumeric, *compare-literal* and *range-literal* must also be alphanumeric. The comparison is based on the greater of the length of *field-name* and *compare-literal* or *range-literal*. The shorter field is padded with spaces to equal the length of the longer field. When *field-name* is numeric, *compare-literal* and *range-literal* must also be numeric and must not have any decimal places.

The set of literal values specified for a given WHEN, including the unspecified values implied by a range, must be unique as compared to the literal values of any other WHEN for the same CASE.

statement-1
statement-n

Statement-1 and *statement-n* represent any number of CA-Easytrieve statements executed when the WHEN comparison is satisfied. Whenever one or more of these statements is a CASE statement, the CASE statements are considered to be nested.

OTHERWISE OTHERWISE is an optional statement that specifies a group of statements to be executed if no WHEN comparison was satisfied. If OTHERWISE is not specified and *field-name* does not equal any of the specified WHEN conditions, execution continues with the statement following END-CASE.

***statement-n*+1** *Statement-n+1* represents any number of CA-Easytrieve statements executed when no WHEN comparisons are equal. Whenever one or more of these statements is a CASE statement, the CASE statements are considered to be nested.

END-CASE END-CASE terminates the body of the CASE statement. END-CASE must be specified after each CASE statement and its associated statements.

Usage Notes

A CASE statement can be nested within a CASE statement. Other conditional execution statements can also be nested within a CASE statement. A CASE statement can be nested within any other conditional execution statement.

Example

The following example uses CASE to analyze the data to select employees' years of service that fall into a range (identified by the WHEN statement) and, as a result, display 'ONE WEEK VACATION' or 'TWO WEEKS VACATION', or for all other cases, display 'THREE WEEKS VACATION'.

```
FILE EMPLOYEE
EMPYRS   5    2   N
...

CASE EMPYRS
  WHEN 0 THRU 4
    DISPLAY 'ONE WEEK VACATION'
  WHEN 5 THRU 10
    DISPLAY 'TWO WEEKS VACATION'
  OTHERWISE
    DISPLAY 'THREE WEEKS VACATION'
END-CASE
...
```

CLOSE Statement

The CLOSE statement closes a file.

Syntax

```
CLOSE file-name
```

Parameters

`file-name`	*File-name* specifies the file to be closed.

Usage Notes

At the termination of each activity, CA-Easytrieve automatically closes all files opened during the activity. You can use the CLOSE statement to close the file before the activity terminates. The next I/O statement using the file re-opens the file.

You can also close an SQL file with the CLOSE statement so that a new cursor can be created. See SQL Database Processing in the *CA-Easytrieve Programmer Guide* for more information.

Note: You cannot use the CLOSE statement to close a printer file or to close an automatic input or output file. Virtual files without RETAIN are deleted when closed. CLOSE has no effect on IDMS files.

Example

```
CLOSE FILEA
```

COMMIT Statement

The COMMIT statement causes a logical unit of work to be established.

Syntax

```
COMMIT
```

Usage Notes

The COMMIT statement establishes the end of the current logical unit of work and the beginning of the next.

The COMMIT statement establishes a recovery point for updates. The ROLLBACK statement can then be used to recover any recoverable actions since the last COMMIT. (The operating environment determines which actions are recoverable. See the *CA-Easytrieve Programmer Guide* for details.)

COMMIT terminates any active holds on files. All open SQL cursors are closed and all updates to databases are committed.

Note: Cursors defined with the HOLD option (DB2 only) are not closed.

Example

```
WRITE PERSNL ADD
. . .
IF . . .
     COMMIT
  ELSE
     ROLLBACK
END-IF
```

Conditional Expressions

Conditional expressions used as parameters of IF and DO statements offer an alternative to the normal top to bottom execution of CA-Easytrieve statements.

Syntax

```
{IF       }             [ {AND}           ]
{DO WHILE }   condition [ {   }  condition]...
{DO UNTIL }             [ {OR }           ]
```

Usage Notes

CA-Easytrieve accepts seven different conditions: Field Relational, Field Series, Field Class, Field Bits, File Presence, File Relational, Record Relational.

Examples

The following are skeletal examples of each type of conditional expression used in an IF statement:

Type	**Example**
Field Relational	IF field-1 = field-2
Field Series	IF field-1 = field-2, field-3, field-4
Field Class	IF field-1 ALPHABETIC
Field Bits	IF field-1 ON X'0F4000
File Presence	IF EOF file-name
File Relational	IF MATCHED file-1, file-2, file-3
Record Relational	IF DUPLICATE file-name

Field Relational Condition

The field relational condition compares fields with values.

Syntax

```
                  Relational
      Subject      Operator        Object
```

```
{IF      }                 {EQ|=         }
{ELSE-IF }                 {NE|¬=|NQ     }  {field-name-2          }
{        }  field-name-1   {LT|< |LS     }  {literal               }
{DO WHILE}                 {LE|<=|LQ|¬>  }  {arithmetic-expression }
{DO UNTIL}                 {GT|> |GR     }
                           {GE|>=|GQ|¬<  }
```

Parameters

Subject *Field-name-1* is the subject of the comparison.

Relational Operator Code any of the *relational operators* to control the condition's evaluation process.

Object Code *field-name-2*, a *literal*, or an *arithmetic-expression* to designate the object of the comparison. Note that alphanumeric literals must be enclosed within single quotes. See the *CA-Easytrieve Programmer Guide* for a description of how CA-Easytrieve evaluates arithmetic expressions.

Alphanumeric Subjects

When the condition subject is an alphanumeric field, the following evaluation rules apply:

1. The object must be either a field or an alphanumeric literal.
2. If necessary, numeric field objects are converted to zoned decimal. Comparison of VARYING alphanumeric fields with numeric fields is not permitted.
3. The comparison is based on the greater of the length of the subject and the length of the object. The shorter item is padded with spaces to the length of the longer item. For downward compatibility with existing CA-Easytrieve programs, this rule is subject to the exception below.
4. When a fixed length subject is compared with a longer fixed length object, the comparison is based on the length of the subject. The object is truncated to match the length of the subject. A warning message is then generated by the compiler.
5. Comparison is logical (bit-by-bit).
6. Comparisons of varying length fields (fields which use the VARYING option of the DEFINE statement) are based on the length of the data at the time of the comparison.

Numeric Subjects

When the condition subject is a numeric field, the following evaluation rules apply:

1. The object must be either a numeric field, a numeric literal, or an arithmetic expression.
2. Comparison is arithmetic.

Mixed Subjects

When the condition subject is a MIXED field, the following evaluation rules apply:

1. CA-Easytrieve only supports equal (EQ =) and not equal (NE ¬= NQ) conditions. If you use any of the other conditional operators, an error occurs.
2. The object must be either a field, an alphanumeric literal, a MIXED literal, or a DBCS literal.
3. CA-Easytrieve does not perform a conversion if the object is an EBCDIC alphanumeric field or literal.
4. If the object is a MIXED field or literal, CA-Easytrieve converts the DBCS portion of data into the DBCS code system of the subject. CA-Easytrieve also converts the shift codes to the values defined for the DBCS code system of the subject in the DBCS Options module.
5. If the object is a DBCS field or literal, CA-Easytrieve converts the data into the DBCS code system of the subject. Once converted, the shift codes defined for the code system of the subject are added to the data.
6. CA-Easytrieve converts numeric field objects to zoned decimal (if necessary).
7. To match the length of the subject, CA-Easytrieve truncates or pads the object. Padding uses the EBCDIC space character. During truncation, no DBCS character is split. When truncation occurs within the DBCS portion of a field, the truncation is adjusted to the nearest double byte boundary.
8. Comparison is logical (bit-by-bit).

DBCS Subjects

When the condition subject is a DBCS field, the following evaluation rules apply:

1. CA-Easytrieve only supports equal (EQ =) and not equal (NE ¬= NQ) conditions. If you use any of the other conditional operators, an error occurs.
2. The object must be either a field, an alphanumeric literal, a MIXED literal, or a DBCS literal.
3. If the object is an EBCDIC alphanumeric field or literal, then CA-Easytrieve converts each character into the DBCS code system of *field-name-1*.

4. If the object is a MIXED field or literal, CA-Easytrieve converts the DBCS portion of data into the DBCS code system of the subject. CA-Easytrieve also converts the EBCDIC portion of data to its equivalent DBCS value based on the code system of the subject. CA-Easytrieve removes shift codes.
5. If the object is a DBCS field or literal, CA-Easytrieve converts the data into the DBCS code system of the subject.
6. If necessary, CA-Easytrieve converts numeric field objects to zoned decimal and then converts the EBCDIC result into the equivalent DBCS characters based on the code system of *field-name-1*.
7. To match the length of the subject, CA-Easytrieve truncates or pads the object. Padding uses the DBCS space character.
8. Comparison is logical (bit-by-bit).

Example

This example illustrates various field relational conditions:

```
FILE PERSNL FB(150 1800)
  EMP#             9    5  N
  EMPNAME         17   20  A
    NAME-LAST  EMPNAME     8  A
    NAME-FIRST EMPNAME +8 12  A
  PAY-NET         90    4  P 2
  PAY-GROSS       94    4  P 2
  SEX            127    1  N
TOTAL-EMP#         W    3  N VALUE 0
TOTAL-SEX          W    3  N VALUE 0
TOTAL-PAY          W    3  N VALUE 0
TOTAL-FIRST-NAME   W    3  N VALUE 0
MALE               W    1  N VALUE 1
JOB INPUT PERSNL NAME MYPROG FINISH FINISH-PROC
   IF EMP# GT 10000
     TOTAL-EMP# = TOTAL-EMP# + 1
   END-IF
   IF SEX NE MALE
     TOTAL-SEX = TOTAL-SEX + 1
   END-IF
   IF PAY-NET LT (PAY-GROSS / 2)
     TOTAL-PAY = TOTAL-PAY + 1
   END-IF
   IF NAME-FIRST EQ 'LINDA'
     TOTAL-FIRST-NAME = TOTAL-FIRST-NAME + 1
   END-IF
*
FINISH-PROC. PROC
   DISPLAY TOTAL-EMP#
   DISPLAY TOTAL-SEX
   DISPLAY TOTAL-PAY
   DISPLAY TOTAL-FIRST-NAME
END-PROC
```

Field Series Condition

The field series condition compares a field to a series or a range of values.

Syntax

```
                             Relational
                Subject       Operator

{IF        }
{ELSE-IF   }                {EQ | =       }
{          }  field-name-1  {             } +
{DO WHILE  }                {NE | ¬= | NQ}
{DO UNTIL  }

   Object

{field-name-2  [     {field-name-3  } ]   }
{              [THRU {              } ]...}
{literal-1     [     {literal-2     } ]   }
```

Parameters

Subject *Field-name-1* is the subject of the comparison.

Relational Operator *Equal* and *not equal* are the only valid relational operators for field series conditions.

Object Code *field-name-2* or a *literal-1* as often as you need to indicate the series of comparison objects. *Field-name-2* THRU *field-name-3*, *field-name-2* THRU *literal-2*, *literal-1* THRU *field-name-3*, or *literal-1* THRU *literal-2* designate a value range. Note that alphanumeric literals must be enclosed within single quotes.

Rules for Evaluation

Evaluation rules for field series conditions are as follows:

1. Alphanumeric (including DBCS and MIXED format fields) and numeric fields are evaluated as in the field relational condition.
2. An equal (=) relational operator tests if the subject is equal to or within range of any of the series of values comprising the object.
3. A not equal (¬=) relational operator tests if the subject is unequal to or outside the range of all the series of values comprising the object.

Note: A comparison within a range is satisfied if the subject is greater than the lesser of the two range values and the subject is less than the greater of the two range values.

Example

This example illustrates the field series condition:

```
FILE PERSNL FB(150 1800)
REGION           1 1 N
BRANCH           2 2 N
DEPT            98 3 N
MARITAL-STAT   128 1 A
*
TOTAL-REGION     W 3 N VALUE 0
TOTAL-BRANCH     W 3 N VALUE 0
TOTAL-DEPT       W 3 N VALUE 0
TOTAL-MARITAL    W 3 N VALUE 0
WORK-REGION      W 2 N VALUE 04
*
JOB INPUT PERSNL NAME MYPROG FINISH FINISH-PROC
   IF REGION = 0, 8, 9
     TOTAL-REGION = TOTAL-REGION + 1
   END-IF
   IF BRANCH NE 01, WORK-REGION
     TOTAL-BRANCH = TOTAL-BRANCH + 1
   END-IF
   IF DEPT EQ 940 THRU 950
     TOTAL-DEPT = TOTAL-DEPT + 1
   END-IF
   IF MARITAL-STAT NE 'M', 'S'
     TOTAL-MARITAL = TOTAL-MARITAL + 1
   END-IF
*
FINISH-PROC. PROC
   DISPLAY TOTAL-REGION
   DISPLAY TOTAL-BRANCH
   DISPLAY TOTAL-DEPT
   DISPLAY TOTAL-MARITAL
END-PROC
```

Field Class Condition

The field class condition determines whether:

- All positions of a field contain alphabetic, numeric, space, or zero characters.
- A nullable field is null.
- The cursor is in a specific field on the screen.
- A field was modified by the terminal user.
- A field is active in a control break of the current report.

Syntax

```
               Subject          Object

                             {ALPHABETIC   }
                             {BREAK        }
                             {CURSOR       }
                             {HIGHEST-BREAK}
{IF        }                 {MODIFIED     }
{DO UNTIL  } field-name [NOT] {NULL        }
{DO WHILE  }                 {NUMERIC      }
                             {SPACE        }
                             {SPACES       }
                             {ZERO         }
                             {ZEROS        }
                             {ZEROES       }
```

Parameters

Subject *Field-name* is the subject of the comparison. Each byte of the field must pass the test before the test is true. The NOT parameter indicates that the condition test is reversed.

Field-name can be indexed or subscripted.

Object The object determines the class of data to be tested for.

{ALPHABETIC} ALPHABETIC tests for the characters A through Z or a blank space in each byte of the subject field.

{BREAK} (Mainframe and UNIX only) BREAK tests whether this field is currently being processed as a CONTROL break field on a report. The BREAK test is an alternative to testing the *field-name* LEVEL for a specific numeric value. *Field-name* must be defined on a CONTROL statement or it must be the reserved word FINAL.

`{CURSOR}` CURSOR tests whether the cursor is in the specified field on the screen. CURSOR can only be used in screen activity procedures.

If CURSOR is used, the condition must refer to a field on a ROW statement within the screen declaration.

Note: Results are unpredictable if:

- *Field-name* contains the ASKIP, PROTECT, or SENDONLY attributes.
- *Field-name* occurs more than once in a screen.
- Two screen fields redefine the same storage area and one of the fields is used in an IF test.
- The subject is indexed or subscripted and the value of the index or subscript has changed since the screen was received.
- The test is performed after the user presses CLEAR, PA1, PA2, or PA3.

`{HIGHEST-BREAK}` (Mainframe and UNIX only) HIGHEST-BREAK tests whether this field caused the CONTROL break on a report. The HIGHEST-BREAK test is an alternative to testing the *field-name* BREAK-LEVEL for a specific numeric value. *Field-name* must be defined on a CONTROL statement or it must be the reserved word FINAL.

`{MODIFIED}` MODIFIED tests whether the terminal operator changed the data in the field. The field is considered MODIFIED only if the contents of the field upon receipt of the screen does not equal the contents of the screen at the time the screen is displayed.

MODIFIED can only be used in screen activity procedures.

If MODIFIED is used, the condition must refer to a field on a ROW statement within the screen declaration.

Note: Results are unpredictable if:

- *Field-name* occurs more than once in a screen.
- Two screen fields redefine the same storage area and one of the fields is used in an IF test.
- The subject is indexed or subscripted and the value of the index or subscript has changed since the screen was received.
- The test is performed after the user presses CLEAR, PA1, PA2, or PA3.

`{NULL}` NULL tests whether a nullable field is NULL.

`{NUMERIC}` NUMERIC tests for the digits 0 through 9 (in the correct format for the field's data type), and for a possible algebraic sign in the low-order position of type P fields or in the high-order position of type N fields.

`{SPACE}` SPACE and SPACES test for the character space in each byte of EBCDIC.

`{ZERO}` ZERO, ZEROS, and ZEROES test for the digit 0 (in the correct format for the field's data type), and for a possible algebraic sign in the low-order position of type P fields or in the high-order position of type N fields.

Example

This example illustrates the use of the field class condition:

```
FILE PERSNL FB(150 1800)
REGION           1    1 N
BRANCH           2    2 N
EMPNAME         17   20 A
  NAME-LAST  EMPNAME     8 A
  NAME-FIRST EMPNAME +8 12 A
*
TOTAL-NUMERIC       W 3  N VALUE 0
TOTAL-NON-ZEROS     W 3  N VALUE 0
TOTAL-ALPHABETIC    W 3  N VALUE 0
*
JOB INPUT PERSNL NAME MYPROG FINISH FINISH-PROC
   IF REGION NUMERIC
     TOTAL-NUMERIC = TOTAL-NUMERIC + 1
   END-IF
   IF BRANCH NOT ZERO
     TOTAL-NON-ZEROS = TOTAL-NON-ZEROS + 1
   END-IF
   IF EMPNAME ALPHABETIC
     TOTAL-ALPHABETIC = TOTAL-ALPHABETIC + 1
   END-IF
*
FINISH-PROC. PROC
   DISPLAY TOTAL-NUMERIC
   DISPLAY TOTAL-NON-ZEROS
   DISPLAY TOTAL-ALPHABETIC
END-PROC
```

Field Bits Condition

The field bits condition compares selected bits of a field for on (1) or off (0) conditions.

Syntax

```
                           Relational
                Subject     Operator    Object

{IF      }                    {ON } {field-name-2}
{DO WHILE} field-name-1 [NOT] {   } {            }
{DO UNTIL}                    ▯OFF} ▯literal     }
```

Parameters

Subject *Field-name-1* is the subject of the comparison. It can be any field type. The NOT parameter indicates the condition test is reversed.

`Relational Operator` The relational operators ON and OFF test for bit values of one or zero respectively.

`Object` *Field-name-2* or a *literal* establish the bit mask to be tested. CA-Easytrieve tests only those bits which correspond to one (1) bits in the mask. The length of the object must equal the length of the subject. When you code *literal* as the object, it must be a hexadecimal literal. Indicate a hexadecimal literal by preceding it with an X and enclosing it in single quotes.

If the subject is a VARYING field, the object must be equal to the length of the data portion of the subject. The test is performed based on the actual length of the subject. The object cannot be a VARYING field.

Example

This example illustrates the use of the field bits condition:

```
DEFINE FIELD-1    W 1  B VALUE X'20'
DEFINE FIELD-NUM  W 4  B VALUE X'FF00FF00'
DEFINE PATTERN-8  W 1  B VALUE X'80'
DEFINE LOWER-CASE W 1  A VALUE X'81'
*
JOB INPUT NULL NAME MYPROG
   IF FIELD-1 ON PATTERN-8
      DISPLAY 'PERFORM CODE FOR PATTERN 8'
   END-IF
   IF LOWER-CASE OFF X'40'
      DISPLAY 'THIS LETTER IS LOWER CASE'
   END-IF
   IF FIELD-NUM  ON X'FF000000'
      DISPLAY '1ST BYTE HIGH VALUES'
   END-IF
STOP
```

File Presence Condition

The file presence condition determines whether a record of the file is currently available for processing.

Syntax

```
                          Subject

{IF      }
{DO WHILE} [NOT]  [EOF] {file-name}
{DO UNTIL}
```

Parameters

Subject *File-name* designates the subject of the test.

Usage Notes

The object of the test is simply the availability of the record for processing. The file is available if the last GET or READ operation was successful and there is a record that can be accessed.

Note: Results are unpredictable if data in a file is referenced after any output operation.

The optional EOF parameter causes the test to be true when the subject is at end-of-file. This test can never be true for automatic input files.

The optional NOT parameter reverses the condition test.

See the *CA-Easytrieve Programmer Guide* for more information on file presence conditions.

Examples

The first example illustrates the use of the file presence condition:

```
FILE PERSNL INDEXED
%PERSNL
*
JOB INPUT NULL NAME MYPROG
  READ PERSNL KEY '00970' STATUS
  IF NOT PERSNL
    DISPLAY '00970 NOT ON FILE'
  ELSE
    DISPLAY EMPNAME
  END-IF
  STOP
```

The next example illustrates the use of the file presence condition in synchronized file processing:

```
FILE PERSNL FB(150 1800)
%PERSNL
FILE INVENT FB(200 3200)
%INVMSTR
FILE SORT1  FB(150 1800) VIRTUAL
COPY PERSNL
FILE SORT2  FB(200 3200) VIRTUAL
COPY INVENT
COUNT-1        W 3 N VALUE 0
COUNT-2        W 3 N VALUE 0
*
SORT PERSNL TO SORT1 USING (ADDR-STATE) NAME MYSORT1
SORT INVENT TO SORT2 USING (LOCATION-STATE) NAME MYSORT2
*
JOB INPUT (SORT1 KEY (ADDR-STATE), +
           SORT2 KEY (LOCATION-STATE)) +
   NAME MYPROG FINISH FINISH-PROC
   IF EOF SORT2
     DISPLAY 'EOF ON SECONDARY'
     STOP
   END-IF
   IF NOT PRIMARY
     DISPLAY 'NO PERSONNEL RECORD- ' LOCATION-STATE
   END-IF
   IF NOT SECONDARY
     DISPLAY 'NO INVENTORY RECORD- ' ADDR-STATE
   END-IF
   IF SORT1
* HOW MANY PERSONNEL RECORDS RETURNED
     COUNT-1 = COUNT-1 + 1
   END-IF
   IF SORT2
* HOW MANY INVENT RECORDS RETURNED
     COUNT-2 = COUNT-2 + 1
   END-IF
*
FINISH-PROC. PROC
   DISPLAY COUNT-1
   DISPLAY COUNT-2
END-PROC
```

File Relational Condition

The file relational condition determines file presence and record matching for more than one file in JOBs with synchronized file input.

Syntax

```
                   Subject

                 [file-name]
IF [NOT] MATCHED [PRIMARY  ] ...
                 [SECONDARY]
```

Parameters

`Subject` The optional *file-name*, PRIMARY, and SECONDARY parameters identify the files to be tested. When you do not code this parameter, the condition is true only if all input files have matching records.

The optional NOT parameter reverses the condition test.

Example

This example illustrates the use of the file relational condition:

```
FILE PERSNL FB(150 1800)
%PERSNL
FILE INVENT FB(200 3200)
%INVMSTR
FILE SORT1  F(150) VIRTUAL
COPY PERSNL
FILE SORT2  F(200) VIRTUAL
COPY INVENT
COUNT-1     W 3 N VALUE 0
*
SORT PERSNL TO SORT1 USING (ADDR-STATE)     NAME MYSORT1
SORT INVENT TO SORT2 USING (LOCATION-STATE) NAME MYSORT2
*
JOB INPUT (SORT1 KEY (ADDR-STATE), +
           SORT2 KEY (LOCATION-STATE)) +
   NAME MYPROG FINISH FINISH-PROC
   IF MATCHED
     COUNT-1 = COUNT-1 + 1
   END-IF
*
FINISH-PROC. PROC
   DISPLAY COUNT-1
END-PROC
```

Record Relational Condition

The record relational condition determines the relationship of the current record of a file to the previous and next records of the same file. This test is valid only for synchronized file processing and single file keyed processing.

Syntax

```
                       Subject

         {DUPLICATE} {file-name}
IF [NOT] {FIRST-DUP} {PRIMARY  }
         {LAST-DUP } {SECONDARY}
```

Parameters

{DUPLICATE} DUPLICATE is true when the previous or next record has the same key as the current record.

{FIRST-DUP} FIRST-DUP is true for the first of two or more records with the same key.

{LAST-DUP} LAST-DUP is true for the last of two or more records with the same key.

Subject The *file-name*, PRIMARY, and SECONDARY parameters identify the file to be tested.

The optional NOT parameter reverses the condition.

Example

This example illustrates the use of the record relational condition:

```
FILE PERSNL FB(150 1800)
%PERSNL
FILE INVENT FB(200 3200)
%INVMSTR
FILE SORT1  FB(150 1800) VIRTUAL
COPY PERSNL
FILE SORT2  FB(200 3200) VIRTUAL
COPY INVENT
COUNT-1     W 3 N VALUE 0
COUNT-2     W 3 N VALUE 0
*
SORT PERSNL TO SORT1 USING (ADDR-STATE) NAME MYSORT1
SORT INVENT TO SORT2 USING (LOCATION-STATE) NAME MYSORT2
*
JOB INPUT (SORT1 KEY (ADDR-STATE) +
           SORT2 KEY (LOCATION-STATE)) +
   NAME MYPROG FINISH FINISH-PROC
   IF DUPLICATE PRIMARY
     COUNT-1 = COUNT-1 + 1
   END-IF
   IF DUPLICATE SORT2
     COUNT-2 = COUNT-2 + 1
   END-IF
*
FINISH-PROC. PROC
   DISPLAY COUNT-1
   DISPLAY COUNT-2
END-PROC
```

CONTROL Statement

The CONTROL statement identifies control fields used for a control report. A control break occurs whenever the value of any control field changes or end-of-report occurs. The control break at end-of-report is equivalent to the final break. A break level is also assigned to each control field. Comparison of control fields is a logical compare.

Syntax

```
        [field-name] [NEWPAGE]
CONTROL [          ] [       ] [NOPRINT] ...
```

```
        [FINAL     ] [RENUM  ]
```

Parameters

`[`*`field-name`*`]`
`[FINAL     ]`

Prior to the first *field-name*, you can code FINAL to specify options for the control break at end-of-report. *Field-name* specifies any non-quantitative field located in an active file or in a W-type working storage field.

Specify control fields in major to minor order.

Note: Varying length, K (DBCS/Kanji), and M (MIXED) fields cannot be specified on a CONTROL statement.

The following three options alter normal processing of a control break:

`[NEWPAGE]` NEWPAGE causes a skip to top-of-page after control break processing is complete for the specified field.

`[RENUM]` RENUM performs the same function as NEWPAGE, and also resets the page number to 1 on the page following the control break.

`[NOPRINT]` NOPRINT suppresses printing the summary line group for the specified control break. All other control break processing for the specified control break is performed as usual.

Usage Notes

You can specify one or more control breaks. If you do not specify any control breaks, a FINAL break is implied.

A break level is assigned to each control field. The system-defined field LEVEL contains the break level used in the BEFORE-BREAK and AFTER-BREAK report procedures. LEVEL has a value of 1 when processing the minor field break. LEVEL contains the number of control fields (n) when processing the major field break. LEVEL contains the number of control fields plus one (n+1) when processing the FINAL control break. The system-defined field BREAK-LEVEL contains the break level of the highest field to break.

An alternative to testing the LEVEL and BREAK-LEVEL fields is to use the IF BREAK and IF HIGHEST-BREAK tests (mainframe and UNIX only). Coding IF BREAK *field-name* is equivalent to coding IF LEVEL = *x*, where *x* is the break level assigned to *field-name*. IF HIGHEST-BREAK performs the same function against the BREAK-LEVEL field. IF BREAK and IF HIGHEST-BREAK have the advantage of dynamically changing the LEVEL value if fields are added to or removed from the CONTROL statement. See the *CA-Easytrieve Programmer Guide* for examples using LEVEL, BREAK-LEVEL, IF BREAK, and IF HIGHEST-BREAK.

Control fields are compared logically, rather than bit-by-bit. For example, packed fields containing zero with a C sign are logically equal to zero with an F sign.

See Report Processing in the *CA-Easytrieve Programmer Guide* for detailed examples of the CONTROL statement.

Example

```
CONTROL FINAL NEWPAGE REGION NEWPAGE BRANCH DEPT
```

COPY Statement

The COPY statement duplicates the field definitions of a named file.

Syntax

```
     {file-name                       }
COPY {                                }
     {[database-file-name]:record-name}
```

Parameters

`file-name` *File-name* is the name of a previously defined file whose fields you want to duplicate.

`[database-file-name]:record-name` *Record-name* is the name of a previously defined database record whose fields you want to duplicate. Optionally, code *database-file-name* for qualification.

Usage Notes

You can code an unlimited number of COPY statements for any one file. CA-Easytrieve duplicates the fields as if they were coded at the place CA-Easytrieve encounters the COPY statement.

The same rules of field definition apply when using the COPY statement (that is, field names must be unique in a given file).

Examples

The following is a COPY statement example:

```
FILE PERSNL  FB(150 1800)
  EMPNAME         17   20 A     HEADING ('EMPLOYEE NAME')
    NAME-LAST  EMPNAME     8 A     HEADING ('LAST' 'NAME')
    NAME-FIRST EMPNAME +8 12 A     HEADING ('FIRST' 'NAME')
FILE SORTWRK FB(150 1800)   VIRTUAL
COPY PERSNL
SORT PERSNL TO SORTWRK USING +
    (NAME-LAST NAME-FIRST) NAME MYSORT
JOB INPUT SORTWRK NAME MYPROG
   PRINT REPORT1
*
REPORT REPORT1
LINE NAME-FIRST NAME-LAST
```

The next example shows a COPY with IDMS:

```
FILE DBASE IDMS(DEMOSS03)
RECORD CUSTOMER 104 KEY(CUST-NO)
  CUST-NO      1 10 A
  CUST-NAME   11 20 A
RECORD SALES   28
  SLS-CUST-NO  1 10 A
FILE DDBASE FB(28 280)
COPY SALES               (fields from RECORD SALES copied)
JOB INPUT (DNASE) NAME MYPROG
   RETRIEVE DBASE +
     SELECT (CUSTOMER AREA 'CUSTOMER-REGION' +
             SALES    ID   'SA' SET 'CUSTOMER-SALES')
   IF PATH-ID EQ 'SA'
     MOVE LIKE SALES TO DDBASE
     PUT DDBASE
   ELSE
     GO TO JOB
   END-IF
```

CURSOR Statement

The CURSOR statement is used within a screen procedure to set the initial position of the cursor in a field for the next display of the screen.

Syntax

```
CURSOR AT field-name
```

Parameters

`field-name` *Field-name* refers to a field on a ROW statement within the screen declaration.

Usage Notes

You can use the CURSOR statement only within screen procedures (AFTER-SCREEN, BEFORE-SCREEN, INITIATION, TERMINATION), or within any procedure performed from a screen procedure.

The CURSOR statement must refer to a field on a ROW statement within the screen declaration.

Note: Results are unpredictable if:

- *Field-name* contains the ASKIP, PROTECT, or SENDONLY attributes
- *Field-name* occurs more than once in screen
- Two screen fields redefine the same storage area and one is used in the CURSOR statement.

Field-name can be subscripted or indexed. However, if the value of the subscript or index changes between the time the CURSOR statement is executed and the time the screen is actually displayed, the CURSOR positioning is ignored. The CURSOR statement:

- Overrides cursor placement if a screen field contains the CURSOR attribute.
- Can be executed any number of times before displaying the screen. The last CURSOR statement executed determines the cursor placement.
- Cannot be moved into an auto-skip (ASKIP) field.

See the *CA-Easytrieve Programmer Guide* for cursor placement hierarchy.

Example

```
SCREEN NAME SCRN1
  ROW 3 WORK-DESCRIPTION
  ROW 5 EMP#
. . .
  BEFORE-SCREEN. PROC
     CURSOR AT EMP#
  END-PROC
```

Chapter

4

Statements D - F

DECLARE Statement

The DECLARE statement enables you to declare named screen attributes and input edit patterns, and to specify how a subprogram is to be linked. Use of declared attributes provides the ability to dynamically change screen attributes during program execution. Use of declared attributes and edit patterns saves you coding time when the set of attributes or edit patterns are used many times.

Syntax

```
             {ATTR [(attribute-list)] }
DECLARE name {PATTERN 'pattern'       }
             {PROGRAM {STATIC|DYNAMIC}}
```

Parameters

name Specify a *name* up to 128 characters for the set of declared screen attributes or set of declared pattern characters.

ATTR [(*attribute-list*)] Specify a list of attribute values. The attribute list must be enclosed in parentheses. See the ATTR Parameter for a list and explanations of valid attributes.

PATTERN '*pattern*' PATTERN enables you to specify a sequence of characters that describe the format of the data in the field. The character string must be enclosed in single quotes.

Note: Use PATTERN to edit complex combinations of data types and character sequences. Use the MASK parameter to edit numeric data.

The valid pattern characters and their meanings are listed below.

Character	Meaning

Character	**Meaning**
A	Represents a lowercase or an uppercase letter
B	Represents a single blank
D	Represents a digit
E	Represents an empty string
L	Represents a lowercase letter
N	Represents an uppercase letter or a national character
U	Represents an uppercase letter
X	Represents any character
"x"	Double quotes surrounding a character or a sequence of characters literally represent the character or sequence of characters contained within. The x represents any character. To literally represent single or double quotes, use two sets of quotes within the surrounding set of double quotes (‘""""‘ or ‘"x""x"‘, ‘"‘‘"‘ or ‘"x’‘x"‘).
blank	Blanks (unless contained in double quotes) serve as delimiters but are otherwise ignored. They can be inserted into the pattern to increase readability.
( )	Represents grouping to control the precedence of operators.
or | or ,	Represents a choice (or alternation operator).
(m) or (m..n) or (m..*) or (*) or *	Represents the repetition of the preceding pattern expression. The m and n represent numbers and m must be less than n. A single number with parentheses indicates the exact number of repetitions. (m..n) represents a range of repetitions, minimum to maximum. An asterisk in a range, (m..*), represents an infinite maximum. An asterisk by itself, (*) or *, represents a range from 0 to infinity.
# or /-/	Represents the remove (or toss) operation. This operation applies only to a single character set at a time and must immediately follow that character set in the pattern. This operation removes the character that matched the character set from the data.
+	Represents character set addition to form another character set.
-	Represents character set difference to form another character set.
concatenation	Concatenation is implied by proximity. For example, DDDU means 3 digits followed by an uppercase letter.

The precedence of operators from highest to lowest:

```
Grouping:                () " "
Set construction:        + -
Actions:        #
Repetition:     (n) (m..n) (m..*) (*)
Concatenation:           proximity
Choice:         |
```

The edit pattern is evaluated from left to right, (the data from the screen is processed from left to right). Patterns examine only one character at a time. They do not look ahead and they do not back track. See the *CA-Easytrieve Programmer Guide* for more information.

PROGRAM {STATIC|DYNAMIC} (Mainframe and UNIX only) PROGRAM enables you to specify how you want to link a subprogram. Specify STATIC to indicate that you want the subprogram to be linked with your CA-Easytrieve program. Specify DYNAMIC to indicate that you want the subprogram to be dynamically loaded. The default is taken from the PARM CALL statement.

Usage Notes

An attribute field can be assigned to another attribute field. Patterns and programs cannot be assigned.

Other than the assignment, DECLAREd screen attributes can only be used on DEFAULT, TITLE, and ROW statements.

Attributes can also be dynamically changed using the SET statement.

Example

```
DECLARE PROTECT-FIELD ATTR (TURQ PROTECT)
DECLARE VARYING-ATTR  ATTR
DECLARE PART-ID       PATTERN 'A"-"DDA'
```

DEFAULT Statement

The DEFAULT statement enables you to specify screen-level overrides of system-defined attributes (Format 1) and message attributes and locations (Format 2).

Syntax

Format 1

```
        {TITLE                                    }
        {FIELD                 {attribute-name  } }
DEFAULT {        [ERROR]  ATTR {                } }
```

```
                  {LITERAL              {(attribute-list)} }
                  {KEY                                     }
```

Format 2

```
                                   {      {attribute-name  } }
                  [INFORMATION]    {ATTR {                 } }
DEFAULT MESSAGE ( [WARNING    ]...) {      {(attribute-list)} }...
                  [ACTION     ]    {                         }
                                   {ROW row-number           }
```

Parameters

`TITLE` Use TITLE to override attributes for all screen titles (fields and literals) in a screen activity.

Note: You can also override attributes at a title item level. See the TITLE Statement.

`LITERAL` Use LITERAL to override attributes for all row literals in a screen activity.

Note: You can also override attributes at a screen item level. See the ROW Statement.

`FIELD [ERROR]` Use FIELD to override attributes for all row fields in a screen activity. Optionally, specify ERROR to override attributes for fields flagged in error by the automatic edit process.

Note: You can also override attributes at a screen item level. See the ROW Statement.

`KEY` Use KEY to override attributes for a function key display area in a screen activity.

```
     {attribute-name  }
ATTR {                }
     {(attribute-list)}
```

Specify either a DECLAREd screen attribute name or one or more attribute keywords. See the ATTR Parameter for a list of attributes. See the DECLARE Statement for how to declare screen attributes.

`MESSAGE` Use MESSAGE to override attributes for any or all message levels (INFORMATION, WARNING, ACTION).

`ROW` ***`row-number`*** Use ROW to override the placement of the message level (INFORMATION, WARNING, ACTION). *Row-number* must be an unsigned integer that does not exceed the maximum screen size (SCREEN ROWCOUNT) and specifies the row number on which the message is displayed.

If ROW is not specified, all messages are displayed one line above the key display area, if used. See the KEY Statement.

Usage Notes

If used, DEFAULT statements must be the first statements coded in a screen activity.

You cannot code overlapping overrides. For example, the following code is in error because the attribute for INFORMATION level messages is coded twice:

```
DEFAULT MESSAGE INFORMATION ATTR BLUE
DEFAULT MESSAGE (INFORMATION WARNING) ATTR GREEN
```

The following attributes are ignored for TITLE, LITERAL, and KEY:

CURSOR
NUMERIC
INVISIBLE
MUSTFILL
MUSTENTER
TRIGGER
ALARM

If coded, CA-Easytrieve issues a warning message during compilation. All of the above attributes are also ignored for MESSAGE, except for ALARM.

Examples

You can use MESSAGE to display INFORMATION level messages in yellow and all other levels of messages in red:

```
DEFAULT MESSAGE INFORMATION       ATTR YELLOW
DEFAULT MESSAGE (WARNING ACTION)  ATTR (RED INTENSE)
```

You can override the placement of messages on a screen using the ROW parameter:

```
SCREEN NAME MENU-SCREEN
  DEFAULT FIELD ATTR (TURQ PROTECT)
  DEFAULT FIELD ERROR ATTR (RED BLINK ALARM)
  DEFAULT MESSAGE (INFORMATION WARNING) ATTR YELLOW  ROW 23
  DEFAULT MESSAGE (ACTION)              ATTR RED     ROW 24
```

DEFINE Statement

The DEFINE statement specifies data fields within a file or within working storage.

Syntax

```
DEFINE +

[file-qualifier:]  field-name +                                     } Field Name

{start-location                                     }               }
{* [+offset-value]                                  }               }
{W                                                  } +             } Location
{S                                                  }               }
{[file-qualifier:] overlay-field-name [+offset-value]}              }

{field-length {A|M|K|N|P|B|U|I|F|S|D} [decimal-positions] [EVEN]}   }
{                                                                } + } Attributes
{[VARYING] [file-qualifier:] model-field-name                   }   }

[UPDATE] +                                                          }
                                                                    }
[HEADING ([#font-number] 'heading-literal' ...)] +                  }
                                                                    }
[INDEX (index-field-name)] +                                        }
                                                                    } Character-
[MASK ({[mask-identifier][BWZ]['mask-literal']|HEX})] +             }   istics
                                                                    }
[OCCURS maximum-occurrences] +                                      }
                                                                    }
[VALUE initial-value]  +                                            }
                                                                    }
[RESET]                                                             }
```

Parameters

DEFINE You can omit the DEFINE keyword for fields defined after the associated FILE statement or for working storage fields defined after any FILE statement. For definitions outside the library, the DEFINE keyword must precede each field definition.

[*file-qualifier*:] *field-name* *File-qualifier*: identifies the appropriate file, record, or working storage for the field you are defining.

Field-name is the name of the field you are defining.

- Can be from 1 to 128 alphanumeric characters in length
- Can contain any character other than a delimiter
- Must begin with A-Z, 0-9, or a national character (#, @, $)
- Cannot be all numeric characters.

Location

You must establish the location of the field's left-most (starting) position in one of the following ways:

{*start-location*} *Start-location* specifies the starting position relative to position one of the current file or record.

Note: *Start-location* must be specified as an unsigned integer.

`{* [+`*`offset-value`*`]}` The * (asterisk) indicates that the field begins in the next available starting position (highest location defined so far, plus 1). The optional +*offset-value* is an offset you want added to the * value. There must be at least one blank between the * and the optional +*offset-value*.

Note: +*offset-value* must be specified as a positive literal.

`{W}` ***or*** **`{S}`** Coding W or S establishes a working storage field; S indicates a static working storage field. CA-Easytrieve spools W fields to report (work) files; it does not spool S fields. See the *CA-Easytrieve Programmer Guide* for more information.

`{[`*`file-qualifier`*`:]` ***`overlay-field-name`*** **`[+`*`offset-value`*`]}`** Specify *overlay-field-name* if you want an overlay redefinition. If you use overlay redefinition, make sure that *field-name* fits within the storage boundaries of *overlay-field-name*. Any indexes associated with *overlay-field-name* also apply to *field-name*.

Specify the optional *file-qualifier*: if the redefined field is in a file or record other than the file or record currently being defined.

The optional +*offset-value* allows you to offset the field from the beginning of *overlay-field-name*.

Attributes

For each *field-name* you define, you must specify the following attributes:

- Field length in bytes
- Data format
- Number (if any) decimal positions
- The optional VARYING parameter for varying length alphanumeric fields.

`{`*`field-length`*`}` *Field-length* specifies the length (in bytes) of the defined field. *Field-length* must be an unsigned integer.

`{A|M|K|N|P|B|U|I|F|S|D}` Specify the data format by selecting one of the following:

- A (alphanumeric)—Use when none of the numeric data types applies to the associated field. A type fields in files are EBCDIC format unless the associated file was declared ASCII on the CODE parameter of the PARM or FILE statement. A type fields in working storage are either EBCDIC or ASCII, depending on the PARM CODE PROCESS value.

- M (MIXED alphanumeric)—(mainframe only) Use when you know the data in the associated field is EBCDIC, DBCS, or a mixture of both. CA-Easytrieve processes the field assuming that it contains EBCDIC data. The DBCS data in this field must be identified by the shift codes in the field's DBCS code system. CA-Easytrieve assumes that the field's DBCS code system is the CA-PSI/DBCS processing code system unless the field belongs to a file that has the CODE parameter specified on its FILE statement. This field type is invalid for those DBCS code systems that do not have an assigned shift code system and cannot support a MIXED field type.
- K (DBCS alphanumeric)—(mainframe only) Use when you know the data in the field is in DBCS format. The length of the field must be a multiple of two. The data in this field is associated with the DBCS code system defined as the CA-PSI/DBCS processing code unless the field belongs to a file that has the CODE parameter specified on its FILE statement.
- N (zoned decimal)—The field contains digits 0 through 9 in external decimal form (for example, 0 = X'F0').
- P (packed decimal)—The field contains numbers that meet IBM's definition of internal packed decimal. For instance, the two-byte packed field containing 123 looks like X'123F'.
- B (binary)—The fields contain binary data. In a quantitative binary *field* (a field with zero or more decimal places specified), the high order bit is the sign bit. In a non-quantitative binary field (a field with no decimal place specification), the high order bit is a binary digit.

  For example, in a one-byte quantitative binary field the following is true:

  ```
  (HEX) 7F = (BIN) 0111 1111 = (DECIMAL) 127

      (HEX) 80 = (BIN) 1000 0000 = (DECIMAL) 128-
  ```

  For a one-byte non-quantitative binary field, the following is true:

  ```
  (HEX) 7F = (BIN) 0111 1111 = (DECIMAL) 127
  (HEX) 80 = (BIN) 1000 0000 = (DECIMAL) 128
  ```

  The table below shows the length equivalent and maximum possible values for signed binary fields:

Field Length in Bytes	Digits	Maximum Value	Minimum Value
1	3	127	128-
2	5	32,767	32,768-
3	7	8,388,607	8,388,608-
4	10	2,147,483,647	2,147,483,648-

The next table shows the length equivalent and maximum possible values for unsigned binary fields:

Field Length in Bytes	Digits	Maximum Unsigned Value
1	3	255
2	5	65,535
3	8	16,777,215
4	10	2,147,483,648

- U (unsigned packed decimal)—Use this parameter for packed data where a sign is not needed. For example, a two-byte unsigned packed field containing 123 looks like X′0123′.
- I (integer)—The field contains integer formatted data in the native format of the host environment. The length of an I field must be two or four bytes. Decimal places must be blank or zero.

  Numeric field capacities for signed I fields (quantitative) are:

Field Length in Bytes	Maximum Value	Minimum Value
2	32,767	-32,768
4	2,147,483,647	-2,147,483,648

Numeric field capacities for unsigned I fields (non-quantitative) are:

Field Length in Bytes	Maximum Unsigned Value
2	65,535
4	4,294,967,295

- F (fixed point ASCII)—(workstation only) The field contains signed ASCII numeric data with or without a decimal point. The sign and the decimal point each use one byte. The maximum length of an F field is 20 bytes. The number of decimal places can be from 0 to 18.

  Numeric field capacities for F fields are:

Field Length in Bytes	Maximum Value	Minimum Value
1	9	0
2	99	-9

Field Length in Bytes	Maximum Value	Minimum Value
19	999,999,999,999,999,999	-999,999,999,999,999,999

- S (single precision)—(workstation only) Use this field type if you need to reference single precision IEEE floating point data in your program. This field must be four bytes in length and can contain zero to seven decimal places.

  Numeric field capacities for S fields are:

Field Length in Bytes	Maximum Value	Minimum Value
4	9,999,999	-9,999,999

  Note: When arithmetic is performed using an S type field:

  - If the source is an S type field, it is converted into a four byte packed decimal temporary field.
  - All arithmetic is performed in packed decimal.
  - If the result is an S type field, the packed decimal result is converted into an S type field.

- D (double precision) - (workstation only) Use this field type if you need to reference double precision IEEE floating point data in your program. This field must be eight bytes in length and can contain zero to fifteen decimal places.

  Numeric field capacities for D fields are:

Field Length in Bytes	Maximum Value	Minimum Value
4	999,999,999,999,999	-999,999,999,999,999

  Note: When arithmetic is performed using a D type field:

  - If the source is a D type field, it is converted into an eight byte packed decimal temporary field.
  - All arithmetic is performed in packed decimal.
  - If the result is a D type field, the packed decimal result is converted into a D type field.

`{[decimal-positions]}` *Decimal-positions* is an option that specifies the desired number of decimal positions for *field-name*. *Decimal-positions* must be specified as an unsigned integer. If *decimal-positions* is specified (even if 0), the field is considered to be quantitative. Otherwise, the field is considered non-quantitative. Quantitative fields are automatically summed on reports. *Decimal-positions* cannot be specified for data type A.

Note: See the table below under Usage Notes for maximum lengths and number of decimal positions for data types.

`[EVEN]` Use EVEN to indicate that a packed decimal field (P) is to contain an even number of digits. The high order digit is zero. For example, a two-byte packed even field can only contain two digits, such as X'012F'.

Note: EVEN is valid only for P fields.

`[VARYING]` Use VARYING to indicate that *field-name* is a varying length field. This means that the length of the data in this field, for each occurrence in separate records, is unique. Varying length fields are alphanumeric and consist of a two-byte length value followed by the data.

Note: The VARYING parameter is not supported for M or K fields.

You can specify VARYING on type A fields. When VARYING is specified, the length attribute (*field-length*) is the total number of bytes the varying length field can occupy (two-byte length plus maximum size of data).

You can specify VARYING for file fields or working storage fields. For file fields the starting position (*start-location*) points to the two-byte indicator. For both file fields and working storage fields, overlay redefinition begins with the two-byte length indicator.

When referencing a VARYING field in your program, *field-name* can be used alone or suffixed as shown below.

Assume *field-name* is 'FLDA':

- FLDA references the entire field (both length and data) as a variable length field
- FLDA:LENGTH references only the length (first two bytes) as a two-byte binary field
- FLDA:DATA references the data portion of the field (from byte three on) as an alphanumeric field.

When a VARYING field is displayed in your output, the data window is based on the maximum length of the field (*field-length* minus two). The length indicator does not display in output unless DISPLAY HEX is specified.

Length restrictions for varying length fields is as follows:

Field Type	Minimum Length	Maximum Length
A	3	32769

The default value for a varying field is a string of zero length. However, if the VALUE option is coded, its value and length become the default for the field.

Note: On the workstation, *field-length* is a two byte integer (I type) if the file for this field has an ASCII code system. If not, *field-length* is a two byte binary field (B type).

`{[`*`file-qualifier`*`:]` ***`model-field-name`*** **`}`** Optionally, you can specify a field name to use as a model for the field you are defining (*field-name*). The attributes used in *model-field-name* are duplicated for *field-name*. If *model-field-name* is in a different file or record, specify the name of that file. If you use this option, you need not specify attributes for *field-name*.

Characteristics

`[UPDATE]` Specify UPDATE for each SQL field to be modified. You can specify UPDATE only for fields defined in a CA-Easytrieve SQL file. Only SQL fields specified as UPDATE can be modified by the UPDATE statement.

If UPDATE is specified on the FILE statement, UPDATE is used for all fields defined in the file.

Note: You must have UPDATE authorization for the column in the SQL table that this file references.

`[HEADING ([#`*`font-number`*`] '`*`heading-literal`*`'...)]` The HEADING option specifies an alternate report heading for *field-name* (the default is the actual *field-name*).

#*Font-number* (mainframe only) defines the number of the font to be used to format *'heading-literal'* when you use *field-name* in a report directed to an extended reporting printer. If the report is directed to a normal printer, CA-Easytrieve ignores this value.

'Heading-literal' specifies an alternate heading and can be up to 128 characters long. *'Heading-literal'* must be enclosed in single quotes. Multiple *'heading-literal'*s, each between single quote marks (' '), enclosed in the parentheses are stacked vertically over the corresponding field when printed.

You can use a unique font number for each *'heading-literal'* by coding the # sign and a value for *font-number* before *'heading-literal'*. Any *'heading-literal'* that does not have a font assigned uses the default font identified in the printer set definition module. See your system administrator for information about default fonts for an extended reporting printer. See the *CA-PSI Subsystems Reporting Environment Guide* for more information about extended reporting.

Note: HEADING can be used in the CA-Easytrieve/Online Screen Painter as a default prompt for *field-name*. See the *CA-Easytrieve/Online User Guide* for details.

`[INDEX (`*`index-field-name`*` ...)]` The INDEX option establishes indexes for *field-name*. You can specify multiple indexes by coding a list of index names (*index-field-name*) enclosed in parentheses.

CA-Easytrieve automatically allocates a four-byte quantitative binary field for each index (two-byte on the Workstation). Any references you make to a field with the INDEX option cause that field's location to be adjusted by the amount contained in *index-field-name*. See the *CA-Easytrieve Programmer Guide* for more information.

`[MASK ({[`*`mask-identifier`*`][BWZ]['`*`mask-literal`*`']|HEX})]` The optional MASK parameter is used to format *field-name* for display.

Any letter from A through Y can be used as an optional *mask-identifier*. You can use the letter to identify a new mask or to retrieve a mask that was previously defined either in the Options Table or by a mask parameter on a previous field definition. If the new mask that you identify does not already exist, CA-Easytrieve retains the mask for future reference. If you subsequently reference *field-name* for display, CA-Easytrieve automatically uses the associated letter identifier to determine the edit mask. Do not use the same identifier to establish more than one mask.

The BWZ (blank when zero) option suppresses the display of *field-name* when it contains all zeros. BWZ can be used by itself or with other options on the MASK parameter.

'Mask-literal' defines an edit mask and must be enclosed within single quotes. The actual edit mask is coded according to the rules specified under the MASK Parameter. See MASK Parameter in the "Statements G–M" chapter.

HEX is a special edit mask that instructs CA-Easytrieve to display the contents of *field-name* in double-digit hexadecimal format. You can display fields of up to 50 bytes with the HEX mask.

Note: HEX edit masks are not allowed for VARYING fields.

`[OCCURS `*`maximum-occurrences`*`]` The OCCURS option establishes an array for *field-name*.

Maximum-occurrences specifies the number of elements in the array (the number of occurrences of *field-name*). *Maximum-occurrence* must be specified as an unsigned integer. You can reference the elements of this array by manipulating the INDEX defined for *field-name* or by using subscripts. See the *CA-Easytrieve Programmer Guide* for more information.

[VALUE *initial-value*] The VALUE option initializes the contents of a field in working storage.

Initial-value can be any valid literal whose type matches the *field-name* type. If *initial-value* is non-numeric, it must be enclosed in single quotes. The maximum length for *initial-value* is 254 bytes.

If the *initial-value* does not match the length of *field-name*, it is truncated or padded according to Assignment rules.

[RESET] Use RESET only for W working storage fields. When you code RESET on the field definition for a W field, RESET returns the field to its initial value whenever a JOB, SCREEN, or SORT is executed. You can use RESET with OCCURS for array fields but not for redefined fields (fields having overlay redefinition). When you use RESET on multiple fields, the fields are reset in the order of the field definitions.

Note: When W working fields are referenced in report processing, a RESET is not performed during the printing of spooled reports.

Usage Notes

DEFINE statements can be automatically generated using the SQL INCLUDE or IDD statements.

Use the following table when specifying *field-length* and *decimal-positions*.

Data Format Code	Maximum Length (bytes)	Number of Decimal Positions
A	32,767*	not valid
K	32,766*	not valid
M	32,767*	not valid
N	18	0 - 18
P	10	0 - 18
B	4	0 - 10
U	9	0 - 18
I	4	0
F	20	0 - 18
S	4	0 - 7
D	8	0 - 15

* For table file fields, ARG (argument) and DESC (description), the maximum length is 254 bytes.

Note: In CICS, the maximum total field length (*field-length* multiplied by *maximum-occurrences*) is 32,759.

See the *CA-Easytrieve Programmer Guide* for complete rules for working with signed (quantitative) and unsigned (non-quantitative) fields.

Examples

The DEFINE statement specifies data fields within a file or within working storage. You usually specify file fields and work fields in your CA-Easytrieve library section, but you can also define them within an activity as the following examples illustrate. The first example shows fields defined in the library:

```
               {  FILE PERSNL  FB(150 1800)
Library        {  DEFINE  EMP#         9    5  N
               {  DEFINE  EMPNAME     17   20  A
               {  DEFINE  EMP-COUNT    W    4  N
                 *
               { JOB INPUT PERSNL NAME MYPROG
               { EMP-COUNT = EMP-COUNT + 1
Activities     { PRINT REPORT1
                 *
               { REPORT REPORT1
               { LINE EMP# EMPNAME EMP-COUNT
```

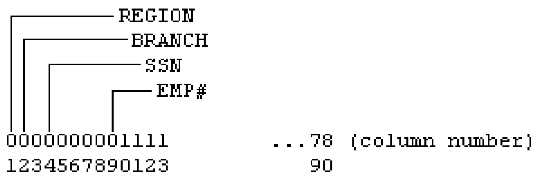
The second example shows fields defined in an activity:

```
               { FILE PERSNL  FB(150 1800)
Library        { SALARY-CODE            134    2 N
               { *

               { JOB INPUT PERSNL NAME MYPROG
               { DEFINE  EMP#              9    5 N
               { DEFINE  EMPNAME          17   20 A
Activities     { PRINT REPORT1
               { *
               { REPORT REPORT1
               { LINE EMP# EMPNAME SALARY-CODE
```

When fields are defined in an activity, each field definition must start with the DEFINE keyword and physically be defined before the field is referenced. In the library section, the use of the DEFINE keyword is optional.

Record Description

The examples below illustrate two ways of describing a record from a personnel file. The first method uses an asterisk (*) to define the starting location of the fields. The second method uses absolute starting positions. In this case, both methods result in the same description. The DEFINE keyword is not needed when the field definitions immediately follow the FILE statement.



```
Method 1
     FILE PERSNL  FB(150 1800)
     REGION           *    1 N
     BRANCH           *    2 N
     SSN              *    5 P
     EMP#             *    5 N
     JOB INPUT PERSNL NAME MYPROG
        PRINT REPORT1
     *
     REPORT REPORT1
     LINE EMP# REGION BRANCH

Method 2
     FILE PERSNL  FB(150 1800)
     REGION           1    1 N
     BRANCH           2    2 N
     SSN              4    5 P
     EMP#             9    5 N
     JOB INPUT PERSNL NAME MYPROG
        PRINT REPORT1
     *
     REPORT REPORT1
     LINE EMP# REGION BRANCH
```

Working Storage Initialization

CA-Easytrieve initializes numeric work fields to zeros and alphabetic work fields to blanks. To initialize these fields to other values, use the VALUE parameter, as shown below:

```
DEFINE CURRENT-MONTH W 10 A VALUE 'JANUARY'
```

Add the RESET parameter to reinitialize the field each time a JOB, SORT, or SCREEN activity is executed.

Varying Length Fields

The VARYING parameter on the DEFINE statement designates varying length fields. An example of a varying length field definition is shown below:

```
FLDA  W  250  A  VARYING
```

Because VARYING is used, this W type work field has two parts that are internally defined as follows:

```
W  2    B 0  for the two-byte field length
W  248  A    for the data
```

Alternate Report Headings

The default report heading for a field is the field name. You can override this default by using the HEADING parameter, as shown in the following example:

```
FILE PERSNL  FB(150 1800)
  EMP#          9    5 N   HEADING('EMPLOYEE' 'NUMBER')
  PAY-NET      90    4 P2  HEADING('NET' 'PAY')
  PAY-GROSS    94    4 P2  HEADING('GROSS' 'PAY')
  WORK-FIELD    W    4 P2  HEADING('AMOUNT' 'OF' 'TAXES')
```

Edit Masks

To add an edit mask to a telephone number, use the MASK parameter:

```
DEFINE PHONE    S    10    N    MASK    '(999) 999-9999'
```

Arrays

The following example defines an array. There are 10 occurrences of the 2-byte numeric field, ELEMENT, in the array. When the array field is used to define the entire array, ARRAY can be used to refer to the entire storage area.

```
DEFINE      ARRAY      W      20    A
DEFINE      ELEMENT    ARRAY  2     N 0    OCCURS 10
```

See the *CA-Easytrieve Programmer Guide* for more information on array processing.

DELETE Statement

The DELETE statement is used to delete a specific row from a CA-Easytrieve SQL file.

Syntax

DELETE [FROM] *file-name*

Parameters

[FROM] Optionally, code FROM for statement readability.

[*file-name*] *File-name* must be the name of a CA-Easytrieve SQL file.

Usage Notes

DELETE performs a DELETE WHERE CURRENT OF cursor. The file must be defined with the UPDATE parameter. See the “SQL Database Processing” chapter in the *CA-Easytrieve Programmer Guide* for more information.

Note: DELETE WHERE CURRENT OF cursor cannot be dynamically processed by the SQL interface for CA-IDMS. To perform SQL deletes, you must code native SQL statements using a searched delete statement.

Example

The following example selects a specific row from the table, then deletes it.

```
FILE PERSNL SQL (PERSONNEL) UPDATE
EMPNAME       *   20  A
WORKDEPT      *   2   P   0
EMPPHONE      *   3   P   0
PROGRAM NAME RETRIEVE-PERSONNEL
  SELECT FROM PERSNL WHERE EMPNAME = 'ROGERS  PAT'
  FETCH FROM PERSNL
  IF EOF PERSNL
    DISPLAY 'EMPLOYEE NOT FOUND'
  ELSE
    DELETE FROM PERSNL
  END-IF
```

DISPLAY Statement

The DISPLAY statement formats and transfers data to the system output device or to a named file. You can code DISPLAY to transfer printed data to the system output device, or you can optionally code a file name after DISPLAY to cause data to be printed to the named file. The DISPLAY statement has three formats; Format 3 can only be used when the file is associated with an extended reporting printer.

Syntax

Format 1

```
           [display-file-name] [{TITLE|NOTITLE}                     ]
   DISPLAY [                 ] [SKIP skip-integer                   ] +
           [SYSPRINT         ] [CONTROL 'carriage-control-character']

           [ [            ]field-name]
           [ [#font-number]          ]
           [ [            ]'literal' ]
           [+offset                  ] ...
           [-offset                  ]
           [COL column-number        ]
           [POS position-number      ]
```

Format 2

```
              [display-file-name] [{TITLE|NOTITLE}                     ]
      DISPLAY [                 ] [SKIP skip-integer                   ] +
              [SYSPRINT         ] [CONTROL 'carriage-control-character']

                  {file-name  }
              HEX {field-name }
                  {record-name}
```

Format 3

```
                    [display-file-name] [                         ]
           DISPLAY  [                 ] [CONTROL 'control-literal']
                    [SYSPRINT         ] [                         ]
```

Parameters

Format 1

```
[display-file-name]
[SYSPRINT         ]
```

When you specify *display-file-name*, CA-Easytrieve prints data to the named file. The named file should be designated as a PRINTER file or unpredictable results can occur. If you do not specify *display-file-name*, the default is SYSPRINT. SYSPRINT implies the system output device. The actual destination of SYSPRINT is determined by the environment or a site option. See your system administrator for more information.

`[{TITLE|NOTITLE}]` The TITLE option specifies that a skip to a new page occurs before the data is printed. It also produces any titles and headings if coded in a report procedure. NOTITLE specifies that a skip to a new page occurs but titles and headings are not produced.

`[SKIP` *`skip-integer`*`]` The SKIP *skip-integer* option specifies the number of lines skipped before printing data. When *skip-integer* is zero, the current line being DISPLAYed overlays the previous line output to *display-file-name*.

[CONTROL '*carriage-control-character*'] The CONTROL *'carriage-control-character'* option sets the print carriage control character for the print line. Valid alphanumeric values for *'carriage-control-character'* are 0 through 9, +, -, A, B, or C. Depending on the make and model of impact printer used, these characters select a precoded channel on a carriage control tape that determines print line positions associated with the form to be printed.

When *display-file-name* is associated with an extended reporting printer, the printer must support ANSI or machine carriage controls. See the *CA-PSI Subsystems Reporting Environment Guide* for more information.

Note: This parameter is not valid for use in REPORT procedures.

[#*font-number*] (Mainframe and UNIX only) #*Font-number* identifies the font that CA-Easytrieve uses for the next display item. You can only specify this option if *display-file-name* has been associated with an extended reporting printer. #*Font-number* identifies the number of a font defined for the extended reporting printer assigned to receive the print output. If you do not code the font index, then the next display item uses the default font for the assigned extended reporting printer.

[*field-name*]
['*literal*']

Code *field-name* or *'literal'* in the order you want them to appear on the printed line.

Note: K fields are not valid on a DISPLAY statement.

[+*offset*]
[-*offset*]

The space adjustment options, +*offset* adds or -*offset* subtract horizontal line spaces preceding the next display item.

Note: ±*Offset* does not extend space beyond the left and right margin set points.

[COL *column-number*] The COL *column-number* option specifies the absolute print column number on which CA-Easytrieve begins to print the next display item. *Column-number* can be any value that does not extend beyond the line margins.

When using an extended reporting printer, an error occurs if two or more fields or literals overlap. See the *CA-PSI Subsystems Reporting Environment Guide* for more information.

[POS *position-number*] The POS *position-number* option coded in a DISPLAY statement within report procedures causes the next display item to be left-justified under the corresponding *position-number* item in the LINE 01 statement.

When using an extended reporting printer, an error occurs if two or more fields or literals overlap. See the *CA-PSI Subsystems Reporting Environment Guide* for more information.

Format 2

```
    {file-name  }
HEX {field-name }
    {record-name}
```

CA-Easytrieve produces a hexadecimal- and character-dump of the current *file-name* or *field-name*, whichever you specify. For CA-IDMS files, *record-name* refers to any record (segment); *file-name* refers to all records (segments).

Note: HEX *file-name* cannot be used in REPORT procedures.

Format 3 (Mainframe and UNIX only)

You can only use this format of the DISPLAY statement when *display-file-name* is associated with an extended reporting printer. A syntax error occurs if *display-file-name* is not an extended reporting printer. See the *CA-PSI Subsystems Reporting Environment Guide*, or the *CA-Easytrieve for UNIX User Guide* for more information about extended reporting.

`[CONTROL 'control-literal']` You can use the CONTROL parameter to output printer control records. *'Control-literal'* can be an alphanumeric or hexadecimal literal that CA-Easytrieve outputs to the print file without paper control information. These control cards contain instructions to extended reporting printers. Print control records for some printing systems define the specification of the font sets that CA-Easytrieve uses for a particular report. These control cards can be output to the print data set before a report that uses the loaded font sets.

Usage Notes

Unless you specify relative or absolute positioning, the first data entry of each DISPLAY statement begins in column 1 of the print line. Each data entry that follows is printed next to the preceding entry. For HEX displays, the output is printed five lines per 100 bytes of the record or field.

When you use DISPLAY in REPORT procedures, output is always in the appropriate place in the report. However, when you use DISPLAY in a JOB activity, the output can be interspersed with the first unSEQUENCED report if no *file-name* is specified.

Data displayed to an output file is in an edited format. DISPLAY is not valid for nullable fields, but DISPLAY HEX is valid.

Examples

Format 1

The following example illustrates the use of Format 1 of the DISPLAY statement.

```
FILE BADKEYS FB(150 1800) TERMINAL
FILE PERSNL INDEXED
%PERSNL
FILE INKEYS
WHO * 5 N
JOB INPUT INKEYS NAME MYPROG
   READ PERSNL KEY WHO STATUS
   IF NOT PERSNL
     DISPLAY BADKEYS 'BAD KEY =' +1 WHO
     GOTO JOB
   END-IF
```

When executed, the statements in this example produce the following output:

```
BAD KEY = 00973
```

Format 2

The following example illustrates the use of Format 2 of the DISPLAY statement.

```
FILE PERSNL INDEXED
%PERSNL
FILE INKEYS
WHO * 5 N
JOB INPUT INKEYS NAME MYPROG
   READ PERSNL KEY WHO
   DISPLAY SKIP 2 HEX PERSNL
```

When executed, the statements in this example produce the following output:

```
CHAR  104   G 01963  7ARNOLD  LINDA       1569 COLONIAL TERR ANEW YORK    NY10012                 @    911
ZONE  FFF21683FFFFF44FCDDDDC44DCDCC4444444FFFF4CDDDDCCD4ECDD4CDCE4EDDD4444DEFFFFF4444444444444403670450FFF
NUMR  1048327C019630071956340039541000000015690363659130359901556086920000581001200000000000000058C045C911
      1...5...10...15...20...25...30...35...40...45...50...55...60...65...70...75...80...85...90...95..100


CHAR    082942        21245140401S   1001101968
ZONE  44FFFFFF44444444FFFFFFFFFFFE444FFFFFFFFFF444444444
NUMR  00082942000000002124514040120001001101968000000000
    101...5...10...15...20...25...30...35...40...45...50
```

DLI Statement

The DLI statement provides controlled input/output of an IMS/DLI database. You can use the DLI statement in conjunction with (or independently of) the automatic input associated with RETRIEVE. You can code the DLI statement at any place in a JOB activity that an input/output statement for any other file can be coded. This statement provides complete control over the creation and maintenance of a database.

Syntax

Format 1

```
                {io-record-name} {'function-literal' }
DLI file-name   {               } {                   }  +
                {io-field-name } {function-field-name}

[                ] [     {'search-value-literal' } ]
[SSANO ssa-number ] [SSA {                        } ] ...
[                ] [     {search-value-field-name} ]
```

Format 2

```
    {CHKP} {'seg-len-literal' }
DLI {    } {                   }  id-field-name  +
    {XRST} {seg-len-field-name}

    [ {'checkpoint-len-literal' }                        ]
    [ {                          }  checkpoint-field-name ] ...
    [ {checkpoint-len-field-name}                        ]
```

Format 3

```
DLI CHKP id-field-name
```

Format 4

```
DLI file-name FOR ACCESS
```

Format 5

```
        {'psb-name-literal' }
DLI PCB {                    }
        {psb-name-field-name}
```

Format 6

```
DLI TERM
```

Parameters

Format 1

`file-name` *File-name* identifies the database being processed. *File-name* is the same as the name coded on the *FILE file-name* statement that identifies the DBD to be processed.

```
{io-record-name}
{               }
{io-field-name }
```

Io-record-name or *io-field-name* identifies the input/output area that is to receive the data. *Io-record-name* must be the same as a corresponding *segment-name* coded on a RECORD statement. *Io-field-name* can only be a working storage field. In either case, the area specified must be large enough to contain the longest segment retrieved from the database.

```
{'function-literal' }
{                    }
{function-field-name}
```

You can specify any IMS/DL/I function code whose parameter requirements conform to Format 1. The function code can be specified as either an EBCDIC alphabetic literal (*'function-literal'*) or an alphanumeric *function-field-name* that contains a four-byte alphabetic code. Valid function codes (for example, GNP) are described in IBM's IMS/DL/I Application Programming publications.

```
[                ] [    {'search-value-literal' }]
[SSANO ssa-number] [SSA {                       }]
[                ] [    {search-value-field-name}]
```

Code the optional SSANO and SSA parameters when the database activity to be performed cannot be satisfied without using segment search arguments. SSANO identifies a *function-field-name* that represents a four-byte binary field. *Function-field-name* can be set dynamically to control the number of SSAs used in database system calls. This value overrides the assumed number which is equal to the count of SSA parameter entries. The SSA parameter supplies segment search argument values. You can code the SSA values as *search-value-field-name* or an alphabetic literal (*'search-value-literal'*). The SSA value must contain the segment search argument in the exact form required by IMS/DL/I. If *search-value-field-name* contains DBCS data, the DBCS code system of *search-value-field-name* must equal the DBCS code system of *file-name*.

Format 2

Use Format 2 of the DLI statement to perform a symbolic checkpoint/restart. You must specify the compatibility option (COMPAT=YES) for the PSB being processed; this option generates a dummy PCB that acts as an I/O PCB during checkpoint/restart processing. Test CHKP-STATUS to determine the results of the call. For more information on CHKP-STATUS, see the "File Processing" chapter in the *CA-Easytrieve Programmer Guide*. See IBM's IMS/DL/I Application Programming publications for more information on symbolic checkpoint/restart.

```
{CHKP}
{    }
{XRST}
```

Code CHKP to perform symbolic checkpoint or XRST to perform a symbolic restart.

```
{'seg-len-literal' }
{                  }
{seg-len-field-name}
```

'Seg-len-literal' or *seg-len-field-name* specifies the length of the longest segment (or path of segments) in the PSB. *'Seg-len-literal'* must be a four-byte binary field.

[*id-field-name*] *Id-field-name* must identify a 12-byte area in working storage. The first eight bytes of this area contain the checkpoint-ID. You should set the 12-byte area to spaces before performing the DLI XRST operation, then test it after performing the operation. If your program is being started normally, the area will still contain spaces. If your program is being restarted from a checkpoint, the area will contain the checkpoint-ID that you supplied during the DLI CHKP operation and in the restart JCL.

Optionally, you can also specify up to seven checkpoint areas in working storage that are saved during each checkpoint and restored during a restart.

```
{'checkpoint-len-literal' }
{                         }
{checkpoint-len-field-name}
```

'Checkpoint-len-literal' or *checkpoint-len-field-name* specifies the length of the checkpoint area defined by *checkpoint-field-name*. *Checkpoint-len-field-name* must be a four-byte binary field.

[*checkpoint-field-name*] *Checkpoint-field-name* must identify a field in working storage. The length of this checkpoint area is specified by *checkpoint-len-field-name* or *'checkpoint-len-literal'*.

A checkpoint can be taken on a maximum of seven areas.

Format 3

Use Format 3 of the DLI statement to perform a basic checkpoint. For IMS, you must specify the compatibility option (COMPAT=YES) for the PSB being processed; this option generates a dummy PCB that acts as an I/O PCB during checkpoint processing. Test CHKP-STATUS to determine the results of the call. For more information of CHKP-STATUS, see the "File Processing" chapter in the *CA-Easytrieve Programmer Guide*. For additional information on basic checkpoint, see IBM's IMS/DL/I Application Programming publications.

CHKP CHKP causes a basic checkpoint to be performed.

id-field-name *Id-field-name* must identify an 8-byte area in working storage. This area contains the checkpoint-ID.

Format 4

Use Format 4 of the DLI statement when you are calling a subprogram (such as a COBOL program) which accesses DL/I records. Coding this statement before referencing DLI fields causes the fields to become available for processing.

DLI *file-name* FOR ACCESS *File-name* refers to a CA-Easytrieve file definition containing the appropriate PCB field, record and record field definitions.

Format 5

Use Format 5 to schedule a PSB. Format 5 is used for CICS execution only. In other environments, it is ignored. Any program that executes under CICS must schedule the PSB before accessing any PSB, including programs that are accessing DL/I with the RETRIEVE statement. For activities that use the RETRIEVE statement, the PSB must be scheduled in a JOB START proc, or in a previously-executed activity. When a PSB is scheduled, it stays scheduled until one of the following occurs:

- Task termination
- Syncpoint
- Execution of the DLI TERM statement (see Format 6).

Test the UIBFCTR and UIBDLTR system-defined fields to determine the results of the operation. See IBM's IMS/DL/I Application Programming publications for more information about scheduling PSBs and the values of UIBFCTR and UIBDLTR.

```
{'psb-name-literal'}
{                  }
{sb-name-field-name}
```

'Psb-name-literal' or *psb-name-field-name* specifies the PSB to be scheduled. The maximum length of a PSB name is eight bytes.

Format 6

Use Format 6 to terminate a PSB. Format 6 is used only for CICS execution. In other environments, it is ignored. Test the UIBFCTR and UIBDLTR system-defined fields to determine the results of the operation. See IBM's IMS/DL/I Application Programming publications for more information about scheduling PSBs and the values of UIBFCTR and UIBDLTR.

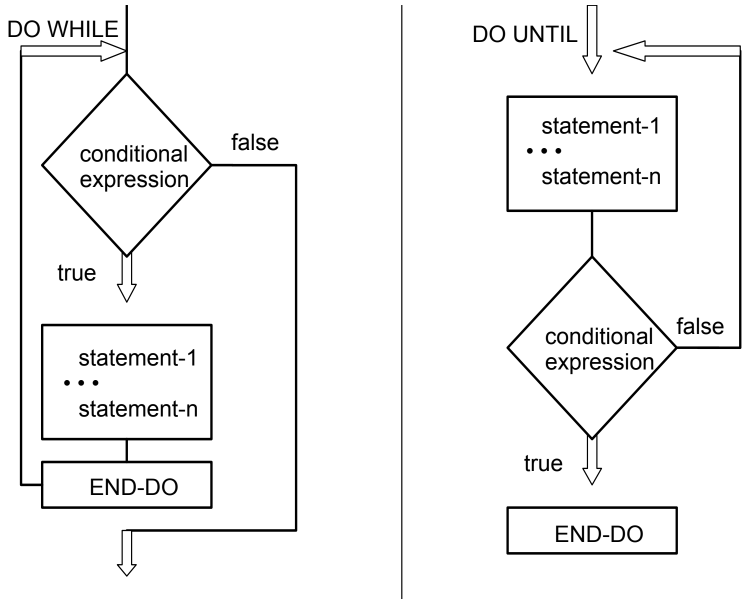
DO and END-DO Statements

The loop control statements DO and END-DO control and delimit repetitive program logic.

Syntax

```
     {WHILE}
DO   {     }     conditional-expression
     {UNTIL}

        statement-1
        ...
        statement-n

END-DO
```

The following diagram illustrates DO and END-DO statement logic:



Parameters

```
{WHILE}
{     }
{UNTIL}
```

A WHILE loop evaluates the condition at the top of a group of statements. The UNTIL loop evaluates the condition at the bottom of a group of statements.

`conditional-expression` Specify the condition that is the basis for the continuing execution of the loop. See Conditional Expressions for conditional expression syntax.

`END-DO` END-DO terminates the body of the loop associated with the DO statement. An END-DO statement must be specified after each DO statement and its associated statements.

Usage Notes

DO WHILE

The truth value of the conditional expression determines whether statement-1 through statement-n are executed. *Statement-1* ... *statement-n* represents any number of CA-Easytrieve statements. When the conditional expression is true, the statements are executed and the program branches back to test the conditional expression. The program continues to loop as long as the conditional expression is true. When the conditional expression is false, the program branches to the statement following END-DO.

DO UNTIL

Statement-1 through statement-n are executed. The truth value of the conditional expression determines whether the group of statements are executed again. When the conditional expression is true, the program branches to the statement following the END-DO. When the conditional expression is false, the program branches back to execute the statements. The program continues to loop until the conditional expression is true.

Examples

DO UNTIL statement example:

```
FILE FILEA
ELEMENT   1   10  A    OCCURS 10
CTR       W    2  N
JOB INPUT FILEA
  CTR = 1
  DO UNTIL CTR > 10
    DISPLAY ELEMENT (CTR)
    CTR = CTR + 1
  END-DO
```

DO WHILE loop nesting example:

```
 DEFINE COUNT-1 W 3 N VALUE 0
 DEFINE COUNT-2 W 3 N VALUE 0
 DEFINE RESULT  W 3 N VALUE 0
 *
 JOB INPUT NULL NAME MYPROG
    DO WHILE COUNT-1 LT 10
      COUNT-1 = COUNT-1 + 1
      COUNT-2 = 0
      DO WHILE COUNT-2 < 10
        COUNT-2 = COUNT-2 + 1
        RESULT = COUNT-1 * COUNT-2
        DISPLAY 'COUNT-1= '  COUNT-1 '  COUNT-2= ' COUNT-2 +
                '  RESULT= ' RESULT
      END-DO
    END-DO
 STOP
```

DRAW Statement

(Workstation only) The DRAW statement produces graphic output by initiating a GRAPH subactivity.

Syntax

```
DRAW [graph-name]
```

Parameters

[*graph-name*] *Graph-name* is the name of the graph as specified on a GRAPH statement. If not specified, CA-Easytrieve assumes *graph-name* is the first graph in the JOB activity.

Usage Notes

All graph output is scheduled for deferred formatting and display, following re-sequencing of the graph data. The data is written to an intermediate file (referred to as a graph work file).

See the “Graph Processing” chapter in the *CA-Easytrieve Programmer Guide* for detailed examples of the use of the DRAW statement in graph processing.

ELEMENT-RECORD Statement (CA-IDMS)

(Workstation only) Code ELEMENT-RECORD statements after a LOGICAL-RECORD statement to identify the element records that comprise the logical record.

Syntax

```
ELEMENT-RECORD   record-name
```

Parameters

record-name *Record-name* is the one-to 16-character name of the element record as defined within the logical record.

Usage Notes

The name of each field must be unique within the element record. However, it is not necessary for the field to be unique within the logical record that contains the element record being defined. If a field defined in another element record has the same name as a field in the named element record, then all references to either field must be qualified with the name of the field's containing element record. See the "CA-IDMS Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

You can use the IDD SUBSCHEMA statement to automatically generate LOGICAL-RECORD, ELEMENT-RECORD, and DEFINE statements.

ENDPAGE Report Procedure

An ENDPAGE procedure can be used to produce page footing information. It is invoked whenever end-of-page is detected.

Syntax

```
ENDPAGE. PROC
```

Usage Notes

A ENDPAGE procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

ENDPAGE is typically used to produce page totals or other annotations, as in the following example of page footer annotation:

```
FILE FILE1
LAST-NAME  1  5 A
STATE      6  2 A
ZIP        8  5 N
PAY-NET    13 5 N 2
JOB INPUT FILE1 NAME MYPROG
  PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65 +
  SUMMARY  SUMCTL DTLCOPY
  SEQUENCE STATE ZIP LAST-NAME
  CONTROL  STATE NEWPAGE ZIP
  TITLE 'REPORT FOR THE STATE OF' STATE
  LINE 01  LAST-NAME STATE ZIP PAY-NET
*
ENDPAGE. PROC
  DISPLAY SKIP 2 '* CONFIDENTIAL - FOR INTERNAL USE ONLY *'
```

```
END-PROC
*
```

END-PROC Statement

The END-PROC statement delimits the statements in a procedure. A procedure is a group of user-written CA-Easytrieve statements designed to accomplish a particular objective.

Syntax

```
END-PROC
```

Usage Notes

See the PROC Statement for more information.

ENDTABLE Statement

ENDTABLE is used to delimit instream data used to create small tables. ENDTABLE must be coded in the first eight positions of the source statement. The ninth position must be blank.

Syntax

```
ENDTABLE
```

Usage Notes

All data required to create an instream table file must be coded between the definition statements and the ENDTABLE statement.

If the table data is to be stored in a macro, you must store the entire table definition from the FILE statement through the ENDTABLE statement.

Example

```
FILE  DAYTABL  TABLE   INSTREAM
   ARG 1    1  A.   DESC   3  9  A
1 SUNDAY
2 MONDAY
 ...
7 SATURDAY
ENDTABLE
```

EXECUTE Statement

The EXECUTE statement invokes a JOB, SORT, or SCREEN activity from either a PROGRAM or SCREEN activity.

Syntax

```
EXECUTE {job-name|sort-name|screen-name}
```

Parameters

`{job-name|sort-name|screen-name}` Name the JOB, SORT or SCREEN activity to be executed.

Usage Notes

The EXECUTE statement transfers control to an activity. After the activity is executed, control returns to the next executable statement following the EXECUTE. You cannot invoke a JOB, SORT or SCREEN activity in a JOB or SORT activity.

EXECUTE statements in a SCREEN activity can invoke other activities. This is called activity nesting. However, recursion is not permitted. That is, ACTIVITY A can EXECUTE ACTIVITY B, but ACTIVITY B cannot then EXECUTE ACTIVITY A.

Note: Recursion cannot be detected in the program. If it is attempted, unpredictable results can occur.

You can use the EXECUTE statement to invoke multiple stacked windows from a SCREEN activity. When each stacked window terminates with an EXIT, the full screen image is restored to the screen image that existed before the EXECUTE. An EXIT from the primary screen does not restore the screen.

Example

```
PARM-FIELD  W  5  A
PROGRAM NAME SAMPLE-PROGRAM USING PARM-FIELD
  IF PARM-FIELD = 'DAILY'
    EXECUTE DAILY-JOB
  ELSE
    EXECUTE WEEKLY-JOB
  END-IF
  JOB NAME DAILY-JOB
  . . .
  JOB NAME WEEKLY-JOB
```

EXIT Statement

The EXIT statement terminates a SCREEN activity.

Syntax

```
EXIT
```

Usage Notes

When an EXIT statement is encountered and the SCREEN activity was invoked from a PROGRAM or SCREEN activity, control is returned to the statement following the EXECUTE statement.

If a SCREEN activity was invoked by an implied PROGRAM activity, EXIT terminates the program.

EXIT is a branch action that can be invoked directly by pressing a particular attention key. See the KEY Statement for more information.

Example

```
SCREEN NAME MENU
KEY F3
...
AFTER-SCREEN. PROC
  IF KEY-PRESSED = F3
    EXIT
  END-IF
END-PROC
```

FETCH Statement

The FETCH statement is used to retrieve rows from a CA-Easytrieve SQL file.

Syntax

```
FETCH [FROM] file-name
```

Parameters

[FROM] Optionally, code FROM for statement readability.

file-name *File-name* is the name of a CA-Easytrieve SQL file.

Usage Notes

The FETCH statement retrieves rows from the open cursor and places the data in the file's data area. If the file does not have an open cursor associated with it, the cursor previously SELECTed is reopened. If no cursor was previously selected, the default cursor (SELECT all defined fields FROM table) is opened.

You cannot use controlled statements (SELECT, FETCH, CLOSE) in a SORT or REPORT procedure.

The FETCH statement cannot reference an automatic input file in the same JOB activity. You can FETCH from a file other than the automatic input file.

Example

Following is an example of a PROGRAM activity that uses a default cursor.

```
FILE PERSNL SQL (PERSONNEL)
EMPNAME       *   20  A
WORKDEPT      *   2   P   0
PROGRAM NAME RETRIEVE-PERSONNEL
  FETCH FROM PERSNL
  DO UNTIL EOF PERSNL
    DISPLAY EMPNAME +2 WORKDEPT
    FETCH FROM PERSNL
  END-DO
```

The above PROGRAM activity simply fetches each row of the table and displays the fields. The DO loop repeats the process until end-of-file.

FILE Statement

FILE statements must describe all files that your program references. You can also access SQL, CA-IDMS, and IMS/DL/I databases using CA-Easytrieve files. Code FILE statements at the beginning of a program after the PARM statement, if one is used. Not all parameters are necessary (or valid) for describing any one file. A review of all parameters will quickly indicate those required for any particular file.

Code the optional parameters and subparameters of the FILE statement in any order following the file-name. As shown, you must code multiple subparameters within parentheses. The complete syntax of the FILE statement is shown below:

Syntax

```
FILE file-name +

  [SEQUENTIAL                              [CREATE [RESET]] ]              }
  [INDEXED                                 [UPDATE         ] ] +           }
  [RELATIVE                                                 ]              }
  [SQL ([owner-name.] table-name [correlation-name]{,}...)  ]              }  File
  [IDMS (subschema-name [RESET])                            ]              }  Type
  [     {relative-position           }                      ]              }
  [DLI ({                            }[RESET])              ]              }
  [     {dbd-name[relative-occurrence]}                     ]              }

  [         {'password-literal' } ]
  [PASSWORD {                   } ] +
  [         {password-field-name} ]

  [NOVERIFY] +

  [                   [        {parm-literal    }      ]            ]
  [EXIT (program-name [USING ( {                } ...) ] [MODIFY]) ]  +
  [                   [        {parm-field-name}       ]            ]

  [CARD                                                             ]      }
  [PUNCH                                                            ]      }Device
  [PRINTER [([PAGESIZE (line-page-size [display-page-size]) +       ]      }Type
  [          [LINESIZE line-length])]                               ]      }

  [F [record-length]                       ]                               }
  [                                        ]                               }
  [V [record-length]                       ]                               }
  [                                        ]                               }
  [  [block-length]                        ]                               }
  [U [FULLTRK     ]                        ]                               }
  [                                        ]                               }
  [FB [ (record-length [block-length] ) ]   +                              }Record
  [   [                [FULLTRK     ]   ]   ]                              }Format
  [                                         ]                              }
  [VB [ (record-length [block-length] ) ]   ]                              }
  [   [                [FULLTRK     ]   ]   ]                              }
  [                                                                        }
  [VBS [ (record-length [block-length] ) ]  ]                              }
  [    [                [FULLTRK     ]   ]  ]                              }

   [WORKAREA area-length] +

  [      [INSTREAM         ] ]
  [TABLE [                 ] ] +
  [      [max-table-entries] ]

  [BUFNO buffers] +

  [DEFER] +

  [ASA] +

  [EXTENDED xrpt-printer]  +

  [CODE {EBCDIC|ASCII|dbcs-code-name}] +

  [KEY key-field-name]  +


  [ [SYSTEM (system-id [PATH path-specification] +           ]
  [                                                          ]
  [         [ACCESS {BTRIEVE   } ]                           ]
  [         [       {DBASE     } ]                           ]
  [         [       {DELIMITED } ]                           ]
  [         [       {HOST      } ]               +           ]
  [         [       {SUPERCALC } ]                           ]
  [         [       {LOTUS     } ]                           ]
  [         [       {RRDS      } ]                           ]
  [                                                          ]
  [         [KEY (key-start-pos key-length)]     +           ]
  [                                                          ]
  [         [INDEX index-file-specification]     +           ]     +
  [                                                          ]
  [         [ACCESS-PATH access-path-number]     +           ]
  [                                                          ]
  [         [CREATE-PATH (access-path-number +           ]   ]
  [         [  KEY (key-start-pos key-length key-type) + ] + ]
  [         [  [DUPLICATE] [MODIFIABLE] ) ...            ]   ]
  [                                                          ]
  [         [OWNER 'owner-id']  +                            ]
  [                                                          ]
[         [CREATE-MODE create-mode-literal]  +            ]
[                                                         ]
[         [PAGESIZE page-size-literal] )                  ]

[        {'file-identifier'         }          ]                        }
[SYSNAME {                          }          ]                        }
[        {file-identifier-field-name}          ]                        }
[                                              ]                        }
[                  [MEMORY]                    ]                        }
[VIRTUAL ([RETAIN] [DISK  ] )                  ]                        }
[                                              ]                        }
[          [   {'terminal-id-literal' } ]      ]                        }
[TERMINAL ([ID {                      } ] +    ]                        }
[          [   {terminal-id-field-name} ]      ]                        }
[                                              ]                        }
[           [NOFORMFEED] +                     ]                        }
[                                              ]                        }
[           [        [BEFORE] ]                ]                        } Data
[           [NEWPAGE [AFTER ] ]... )           ]                        } Set
[                                              ]                        } Type
[        [      {'spool-class-literal' } ]     ]                        }
[SPOOL ( [CLASS {                      } ] +   ]                        }
[        [      {spool-class-field-name} ]     ]                        }
[                                              ]                        }
[        [     {'destination-literal' } ]      ]                        }
[        [NODE {                      } ] +    ]                        }
[        [     {destination-field-name} ]      ]                        }
[                                              ]                        }
[        [       {'user-id-literal' } ]        ]                        }
[        [USERID {                  } ] )      ]                        }
[        [       {user-id-field-name} ]        ]                        }
```

Parameters

`file-name` *File-name* is a one to 128-character name used to define the file to CA-Easytrieve. All statements that operate on the file refer to this name. Every FILE statement must have a *file-name* immediately following the FILE keyword. *File-names* must be unique in the program (that is, the programmer can use a given *file-name* for only one file). The first three characters of *file-name* must be different from the value of the work data set name prefix specified in the Site Options Table (normally EZT).

See the SYSNAME parameter for the relationship between a FILE statement and an external data set.

File Types

CA-Easytrieve processes all standard file types available in the operating environment. On a mainframe, this includes QSAM (Queued Sequential Access Method), VSAM (Virtual Storage Access Method), SQL, printer files directed to an online printer, terminal, or to the operating system spooling subsystem (JES2 or JES3 in MVS and POWER in VSE), CA-IDMS and IMS/DL/I database files, and the CA-Easytrieve Virtual File Manager (VFM). On a workstation, these include fixed and variable length sequential, ISAM (Indexed Sequential Access Method), relative, random, VFM, SQL, and CA-IDMS database files. On UNIX, these include fixed length sequential, variable length sequential (new-line delimited), relative, VFM, SQL, and CA-IDMS database files. If you do not specify a file type, CA-Easytrieve assumes the file is sequential.

`[SEQUENTIAL]` SEQUENTIAL designates a QSAM or VSAM Entry Sequenced Data Set (ESDS) on a mainframe. On a workstation, the file can be a fixed or variable (CR/LF) file. On UNIX, the file can be a fixed length or a variable length (new-line delimited) file. SEQUENTIAL is the default file type if a file type is not specified.

Note: When SEQUENTIAL is specified, FILE-STATUS (a system-defined field) is available for the file.

`[INDEXED]` INDEXED designates a VSAM Key Sequenced Data Set (KSDS), a workstation ISAM file, or a UNIX ISAM file.

`[RELATIVE]` RELATIVE designates a mainframe VSAM Relative Record Data Set (RRDS), or a relative file on a workstation or on UNIX.

`[SQL] ([`*`owner-name`*`.]]` ***`table-name`*** **`[`*`correlation-name`*`] {,} ...)`** The SQL parameter designates an SQL file. This parameter enables CA-Easytrieve to manage the SQL cursor.

Note: Only UPDATE, DEFER, and CODE are valid FILE statement parameters for an SQL file.

Table-name is the name of the SQL table to be accessed. Optionally, qualify the table with *owner-name*. *Table-name* must be enclosed in parentheses.

Correlation-name is the name used to clarify or simplify the table to which a column belongs. If you specify *owner-name* and your DBMS does not permit more than one level of qualification (CA-IDMS, CA-Ingres, or ORACLE), you should code a *correlation-name* for each table.

Note: The comma is a required separator.

When you specify SQL, the file can be used as the subject of either automatic input or controlled processing using the SELECT, FETCH, CLOSE, INSERT, UPDATE, and DELETE statements. These statements support full read-write access to the tables but you do not have to declare, open, and close cursors to manage the tables. Multiple tables for a single file are joined for inquiry only.

`[IDMS (`*`subschema-name`*` [RESET])` IDMS designates the file as a CA-IDMS database file.

Subschema-name is a one to eight-character name that specifies the subschema to be processed.

The optional RESET subparameter requests that all records under control of RETRIEVE be reset to binary zero immediately prior to retrieving each root record.

When the first IDMS FILE statement is encountered, a CA-IDMS Communications Block is created in working storage.

See the "CA-IDMS Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

Note: Fields cannot be defined in association with the IDMS FILE statement. Fields are defined following the RECORD or ELEMENT-RECORD statement.

```
     {relative-position            }
DLI ({                             } [RESET])
     {dbd-name[relative-occurrence]}
```

(Mainframe only) DLI designates the file as an IMS/DL/I database file.

Relative-position is a positive numeric literal that identifies the relative position of the PCB within the PSB to be processed.

Dbd-name specifies the name of the DBD. *Relative-occurrence* is the relative occurrence of like-name DBDs in the PSB. It is only required if two or more DBDs have the same name.

The optional RESET subparameter requests that all records under control of RETRIEVE be reset to binary zero immediately prior to retrieving each root record.

See the "IMS/DL/I Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

`[CREATE [RESET]]` Use CREATE to load the associated file.

(Mainframe only) If you specify RESET, CA-Easytrieve overwrites an existing data set with the REUSE attribute. If RESET is not specified, or if RESET is specified and the associated VSAM data set does not have the REUSE attribute, then CA-Easytrieve receives an error condition when the OPEN is executed.

Note: In CICS, the use of RESET results in an execution error.

(UNIX only) If you specify RESET, CA-Easytrieve deletes an existing data set before creation. If RESET is not specified, new records are appended.

`[UPDATE]` Use UPDATE to permit the file to be updated.

For SQL, code UPDATE to specify that all columns defined for the file can be updated. If you do not specify UPDATE for an SQL file, only those columns defined with the UPDATE parameter can be updated. UPDATE is required to DELETE from or INSERT to an SQL file.

Note: You must have UPDATE, INSERT, or DELETE authorization for the SQL table that this file references.

```
[         {'password-literal' } ]
[PASSWORD {                   } ]
[         {password-field-name} ]
```

'Password-literal' is the optional password for the file. You can specify the password as an alphabetic literal or a hexadecimal quoted literal.

Password-field-name is a field you define that contains the password for the file. CA-Easytrieve accesses the value in *password-field-name* when the file is opened. Any valid password is accepted.

`[NOVERIFY]` Code NOVERIFY to ignore a VSAM open error code of 116(X'74'). This error indicates that a previous job terminated without properly closing the associated VSAM data set. It could also indicate that a job executing on another CPU is using the associated VSAM data set.

CAUTION: Indiscriminate use of NOVERIFY can cause loss of data records.

`[EXIT]` EXIT invokes a user written program for each CA-Easytrieve operation on the file. EXIT is not valid for VFM, SQL, DL/I, or IDMS.

`program-name` Specify the name of the user program.

```
[        {parm-literal   }      ]
[USING ( {               } ...) ]
[        {parm-field-name}      ]
```

USING appends the associated parameters (*parm-literal* or *parm-field-name*) to the standard parameter list passed to the exit program. Field names must be working storage or system defined fields and must be defined in the library section. There is a limit of 62 fields that can be passed to the exit program.

`[MODIFY]` MODIFY specifies that CA-Easytrieve provides input or output services, but that the exit can inspect and modify each record after input and before output.

Device Types

The optional parameters CARD, PUNCH, and PRINTER specify the device type for SEQUENTIAL files. For MVS, if you do not specify one of these parameters, the device type is determined by your JCL.

`[CARD]` (TSO and CMS only) The CARD option retrieves the file data from the system input stream (SYSIN). Only one file in a CA-Easytrieve execution can use the CARD option. Files using this option must be 80-character unblocked records.

Note: In TSO and CMS, you cannot use a CARD file if you want to execute your program interpretively.

(Workstation only) The CARD option indicates the file is a variable length file (carriage-return, line-feed delimited) using an 80-character record buffer.

(UNIX only) The CARD option indicates the file is "stdin." It is treated as a variable length file (new-line delimited) with a maximum record length of 256.

`[PUNCH]` (TSO and CMS only) The PUNCH option indicates punched card output. Files created with this option are 80-character unblocked records.

`[PRINTER]` PRINTER indicates that the file receives printed output routed to either a file, online printer, terminal, or to a subsystem (JES/POWER) data set. Although normal input/output statements (GET, PUT, READ, WRITE) cannot reference PRINTER files, the DISPLAY statement and the PRINTER parameter of the REPORT statement can reference PRINTER files.

`[PAGESIZE (`*`line-page-size`*` [`*`display-page-size`*`])` Specify PAGESIZE to define the logical print length of a printed page. See the REPORT Statement for complete rules for specifying PAGESIZE.

`[LINESIZE `*`line-length`*`]` Code the LINESIZE parameter to specify the maximum number of data characters that can be printed on a line. *Line-length* must be an unsigned integer from 1 to 32767.

Line-length must be at least one less than the length of the data portion of the file's logical record. If the FILE definition does not provide the file's format and logical record length, then no compile time verification of the *line-length* is done.

Line-length provides the default value for any REPORT specifying this file as its PRINTER file.

The default value of LINESIZE is calculated as one less than the data portion of the logical record if the file format and record length are known at compile time. Otherwise, the default is taken from the LINESIZE site option.

There are additional control characters (forms control information) that also must be stored in a logical record. If one of the record format parameters is specified, it must be large enough to hold both the forms control information and the data characters. The value of *line-length* must be less than or equal to the maximum record length minus the size of the forms control information.

Record Format

You can optionally code the record format of non-VSAM files for MVS programs. If you do not code a record format in MVS, CA-Easytrieve obtains it from the operating system when the file is opened. For input files, CA-Easytrieve always obtains the record format from MVS. The record format and length are required for VSE and workstation FILE statements.

```
[F [record-length]                          ]
[                                           ]
[V [record-length]                          ]
[                                           ]
[U [block-length]                           ]
[  [FULLTRK     ]                           ]
[                                           ]
[FB [ (record-length [block-length] ) ]     ]
[   [                [FULLTRK     ]   ]     ]
[                                           ]
[VB [ (record-length [block-length] ) ]     ]
[   [                [FULLTRK     ]   ]     ]
[                                           ]
[VBS [ (record-length [block-length] ) ]   ]
[    [                [FULLTRK     ]   ]   ]
```

CA-Easytrieve supports fixed (F), variable (V), and undefined (U) formats. Fixed and variable length records can be blocked (FB,VB).

[VBS] (MVS only) MVS systems can process Variable Block Spanned (VBS) records using BFTEK=A processing.

[record-length] *Record-length* specifies the maximum record length.

[block-length] *Block-length* specifies the file's maximum block length.

For mainframe variable format files, allow four bytes of the record length for the Record Description Word (RDW) and, if the file is blocked, four bytes of the block size for the Block Description Word (BDW).

Note: To obtain an MVS/DFP system determined block size within CA-Easytrieve on the mainframe, you can do one of the following:

- Include the DSORG, LRECL and RECFM parameters in the original JCL or TSO dynamic allocation. This forces SMS to establish the block size before CA-Easytrieve gets control in OPEN processing. This is only applicable for disk data sets.

  OR

- Define a value of zero for the block length value. If you want CA-Easytrieve to pick up the logical record length from your JCL, code a zero for record-length. You must also code a BLKSIZE=0 in your JCL or code no BLKSIZE parameter at all.

Examples:

```
FILE file-name FB(0 0)
```

This tells CA-Easytrieve to pick up the LRECL from the JCL and to utilize the block size set by SMS.

```
FILE file-name FB(150 0)
```

This tells CA-Easytrieve to pick up the LRECL from this definition and to utilize the block size set by SMS.

```
FILE file-name FB(150 3000)
```

This tells CA-Easytrieve to pick up this definition and ignore both the JCL and the SMS-determined block size.

Note: If you code a zero block size within CA-Easytrieve and/or in your JCL, and your data set is not SMS managed, your program will abend with a 013 open problem.

[FULLTRK] A block length designation of FULLTRK establishes an output block size that equals the maximum track capacity of the direct access device, or the next lower multiple of record size for FB files.

[WORKAREA *area-length*] The WORKAREA option establishes the number of bytes to be allocated as a workarea for the file. WORKAREA cannot be coded if the CARD parameter is specified. *Area-length* specifies the number of bytes to be allocated and must be large enough to contain the longest record processed.

WORKAREA allows you to reference the fields in a file prior to the normal allocation of a file's data buffer. See the "File Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

Note: WORKAREAs are not initialized by CA-Easytrieve.

[TABLE] The TABLE option declares the file as the source for a SEARCH statement to access a table.

Note: VARYING length fields cannot be used for TABLE files.

```
[INSTREAM         ]
[max-table-entries]
```

The INSTREAM option indicates that the table file immediately follows the file description. The size of an INSTREAM table is limited only by the amount of available memory. *Max-table-entries* specifies the maximum number of entries in an external table. If INSTREAM or *max-table-entries* is not specified, the file is an external table whose maximum number of entries is limited by the site option TBLMAX.

[BUFNO *buffers*] BUFNO establishes the number of buffers allocated for the file. *Buffers* can be 1 through 255 for MVS programs. The default value is obtained from the Site Options Table. BUFNO is ignored on the workstation.

`[DEFER]` Coding the DEFER option instructs CA-Easytrieve to delay the opening of the file until the first input or output operation for the file occurs. The default opens all referenced files at the beginning of each CA-Easytrieve activity.

`[ASA]` (MVS and UNIX only) For MVS, the optional ASA parameter sets the DCB A option for RECFM. ASA is ignored on the workstation.

For UNIX, the optional ASA parameter causes output records to be written using the set of mainframe ASA characters in column one. Without ASA, all records are formatted using ASCII printer control characters.

`[EXTENDED `*`xrpt-printer`*`]` (Mainframe and UNIX only) The EXTENDED parameter indicates that the file is to be associated with an extended reporting printer. This means that input/output statements (GET, PUT, READ, WRITE) cannot reference these printer files. However, the DISPLAY statement and REPORT statements can reference these printer files. Unless you code them, record length and blocksize default to those defined for the printer in the printer set definition module.

Xrpt-printer identifies the extended reporting printer whose characteristics are to be associated with this file. You must define the *xrpt-printer* in the printer set definition module.

See the *CA-PSI Subsystems Reporting Environment Guide* or the *CA-Easytrieve for UNIX User Guide* for more information.

`[CODE {EBCDIC|ASCII|`*`dbcs-code-name`*`}]` (Workstation only) Use CODE to define the processing code system (EBCDIC or ASCII) CA-Easytrieve uses for all fields in this file.

If not specified, the default is taken from the program's CODE parameter on the PARM statement. If not specified on the PARM statement, the default is taken from the CA-Easytrieve/Workstation Site Options.

(Mainframe only) Use CODE *dbcs-code-name* to define the DBCS code system to be used for all K and M fields for this file. If not specified, the default is taken from the CODE parameter on the PARM statement for this program. If not specified on the PARM statement, then the default is taken from the processing code system as defined in the CA-PSI Subsystems DBCS Options table.

`[KEY `*`key-field-name`*`]` Use the KEY parameter to specify the key field for your UNIX ISAM file. CA-Easytrieve uses this parameter only when the file is created. After the file is created, this parameter is ignored and key information is obtained from the access method. If KEY is not specified during creation, the first field defined in the file is used as the key field.

[SYSTEM (*system-id*) (Workstation only) Code the SYSTEM parameter to specify information that is specific to a particular operating system or environment. The SYSTEM parameter is ignored on the mainframe and in UNIX.

Use *system-id* to specify the operating system or environment. *System-id* must be PC for the workstation.

[PATH *path-specification*] Use the PATH parameter to specify the physical filename to be associated with this file. *Path-specification* must be a valid physical file name for the operating environment. You can optionally enclose *path-specification* in single quotes if spaces or special characters exist in the physical file name.

Path-specification overrides the SYSNAME *file-identifier* when the program is executed on the workstation. This enables you to code the external data set name for execution on the mainframe and the *path-specification* for the workstation.

```
[ACCESS {BTRIEVE   } ]
[       {DBASE     } ]
[       {DELIMITED } ]
[       {HOST      } ]
[       {SUPERCALC } ]
[       {LOTUS     } ]
[       {RRDS      } ]
```

Code the ACCESS parameter to specify the access mode of file type to be associated with this file.

{BTRIEVE} Specify BTRIEVE if the file is a BTRIEVE file. BTRIEVE is valid only if the file type is INDEXED.

Note: The BTRIEVE Manager must be loaded before you can run a program that accesses a BTRIEVE file.

{DBASE} Specify DBASE if the file is a dBASE III, IV, or 5.0 (.DBF) file. DBASE is valid only if the file type is SEQUENTIAL or unspecified and the record format is fixed.

Note: CA-Easytrieve reads only dBase data files. dBase indexed files cannot be accessed.

{DELIMITED} Specify DELIMITED if the file is a comma-delimited file. DELIMITED is valid only if the file type is SEQUENTIAL or unspecified and the record format is variable.

{HOST} Specify HOST if the file is to be accessed using the IBM High Level Applications Program Interface (HLLAPI). HOST is valid only if the file type is SEQUENTIAL or unspecified.

To use this access mode, you must:

- Specify the HOST transfer operating system (CMS or TSO) and the correct emulator type in the options table. Ensure that you have compiled your program with the desired options in effect.
- Have the appropriate hardware connection the your mainframe.
- Have the mainframe emulator resident or ready for use.
- Have SEND.EXE in your current directory.
- Have the High Level Applications Program Interface (HLLAPI) resident or available for use.
- Have IND$FILE on your mainframe ready to accept workstation file transfer commands.
- OS/2 only—define the .DEF file for the program accessing the HOST.

`{SUPERCALC}` Specify SUPERCALC if the file is a CA-SuperCalc 5 (.CAL) file. SUPERCALC is valid only if the file type is SEQUENTIAL or unspecified and the record format is fixed.

`{LOTUS}` Specify LOTUS if the file is a Lotus 123 (.WKS or .WK1) file. The default access mode for a Lotus file is .WKS format. To override the default, include the extension of the file to be accessed in the PATH parameter if the file is a .WK1 file.

LOTUS is valid only if the file type is SEQUENTIAL or unspecified and the record format is fixed.

`{RRDS}` Specify RRDS if you coded VS as the file type and you want the workstation file to function as a VSAM RRDS file. A VSAM RRDS file is then emulated instead of a KSDS file. Use this parameter to maintain portability between the workstation and the mainframe. See Usage Notes for more information.

RRDS is valid only if the file type is RELATIVE or VS.

`[KEY (`*`key-start-pos key-length`*`)]` Use the KEY parameter to specify the start position and length of the key for your indexed (workstation ISAM) file. Use this parameter only when the file is initially created and then ignored. If you do not code KEY, the first field defined in the file is used as the key field.

KEY can only be coded once for each indexed (workstation ISAM) file, and is ignored if the file is not an indexed file.

Note: Workstation ISAM file keys are always stored in character format and the key length cannot exceed 100 bytes in length. Only alphanumeric and zoned numeric data should be used for the key on the workstation.

[INDEX *index-file-specification*] Use the INDEX parameter to specify the index file name to be used by the ISAM handler. *Index-file-specification* must be a valid file name for the target operating system, and, optionally, can be enclosed in single quotes if spaces or special characters exist in the physical file name. If you do not code INDEX, the index filename is filename.IDX.

INDEX can only be coded once for each indexed (workstation ISAM) file, and is ignored if the file is not an indexed file.

[ACCESS-PATH *access-path-number*] Use the ACCESS-PATH parameter to specify the access path used for record retrieval in a BTRIEVE file. The access path you specify defines the access path for the JOB INPUT, GET, and READ statements.

Access-path-number must be an integer from 0 to 23 and must correspond to an access path that currently exists in the BTRIEVE file. If an access path is not coded, an access path of 0 is used.

ACCESS-PATH can only be coded once for each BTRIEVE file and is ignored if the file is not a BTRIEVE file. See your *BTRIEVE Reference Manual* for more information.

```
[CREATE-PATH (access-path-number  +
   KEY (key-start-pos key-length key type)  +
   [DUPLICATE] [MODIFIABLE]) ...]
```

Use the CREATE-PATH parameter to specify the access paths to be created for record operations in a BTRIEVE file. The access paths you specify are created for use on the JOB INPUT, GET, and READ statements.

You can code multiple CREATE-PATH parameters to create multiple access paths for a single BTRIEVE file. CREATE-PATH is ignored if the file is not a BTRIEVE file. See your *BTRIEVE Reference Manual* for more information.

Access-path-number must be an integer from 0 to 23. Each *access-path-number* you specify must be unique.

Use the KEY parameter to specify the start position, length, and type of key to be created in a BTRIEVE file. *Key-start-pos* and *key-length* must be integers. *Key-length* cannot exceed 255 bytes. *Key-type* can be any valid CA-Easytrieve data type.

CA-Easytrieve data types are mapped to BTRIEVE key types as follows:

CA-Easytrieve data type	BTRIEVE key type
A	STRING
N	STRING
P	DECIMAL
B	STRING
U	STRING
I	INTEGER
F	STRING
S	FLOAT
D	FLOAT

Use DUPLICATE if you want to permit duplicate keys for this access path.

Use MODIFIABLE if the contents of the key can be changed during a WRITE UPDATE operation.

`[OWNER '`*`owner-id`*`']` Use the OWNER parameter to restrict access to your BTRIEVE file. *'Owner-id'* must be an alphanumeric literal up to eight characters in length. *'Owner-id'* is required to access your BTRIEVE file after it has been created.

`[CREATE-MODE` ***`create-mode-literal`*****`]`** Use CREATE-MODE to set access rights and encryption to be performed on a BTRIEVE file. This parameter is ignored if the OWNER parameter is not coded.

Create-mode-literal must be an integer from zero to three and is defined as follows:

create-mode-literal	Description
0	Requires an owner ID for any file access (no data encryption)
1	Permits read-only access without an owner ID (no data encryption)
2	Requires an owner ID for any file access (with data encryption)
3	Permits read-only access without an owner ID (with data encryption)

Note: If the CREATE-MODE parameter is not coded and the OWNER parameter is coded, the file is created with CREATE-MODE 0.

`[PAGESIZE` ***`page-size-literal`*** **`]`** Use the PAGESIZE parameter to set the data page size used by BTRIEVE to create the file.

Page-size-literal must be an integer from 512 to 4096 and must be a multiple of 512. CA-Easytrieve defaults to a page size of 2560.

The value you select for *page-size-literal* can have performance and disk space implications. See your *BTRIEVE Reference Manual* for more information.

Data Set Types

```
SYSNAME  {'file-identifier'         }
         {                          }
         {file-identifier-field-name}
```

Code SYSNAME to associate a CA-Easytrieve file with an external data set. *'File-identifier'* must be an alphanumeric string. *File-identifier-field-name* must be defined as an alphanumeric field of the required length. CA-Easytrieve accesses the value of *file-identifier-field-name* when the file is opened. The required length and the set of valid characters depend on the file type, the implementation, and the operating system.

Note: If SYSNAME is not specified, then the *file-name* specified after the FILE keyword is used. The length of *file-name* must conform to operating system standards.

CICS:

- For file types SEQUENTIAL, INDEXED, and RELATIVE, this is the FCT name of the associated VSAM data set. The requirements for the format of the *file-identifier* character string are the same as the requirements for the FCT name. If fewer than eight characters are provided, the value is padded with blanks on the right to obtain a string of eight characters. *File-identifier-field-name* must be an eight-byte sequential alphanumeric field of any format. Only uppercase alphabetic and numeric digits are allowed. The first character must not be a digit.
- For device type PRINTER, the SYSNAME parameter cannot be used. An execution error occurs.

TSO:

- For file types SEQUENTIAL, INDEXED and RELATIVE, and device type PRINTER, this is the DDname of the associated data set. The requirements for the format of the *file-identifier* character string are the same as the requirements for the DDname. If fewer than eight characters are provided, the value is padded with blanks on the right to obtain a string of eight characters. *File-identifier-field-name* must be an eight-byte alphanumeric field of any format. Only uppercase alphabetic and numeric digits are allowed. The first character must not be a digit.
- For data set type VIRTUAL, the SYSNAME parameter is ignored.

CMS:

- For file types SEQUENTIAL, INDEXED and RELATIVE, and device type PRINTER, this is the FILEDEF or DLBL name of the associated data set. The requirements for the format of the *file-identifier* character string are the same as the requirements for the FILEDEF or DLBL name. If fewer than eight characters are provided, the value is padded with blanks on the right to obtain a string of eight characters. *File-identifier-field-name* must be an eight-byte alphanumeric field of any format. Only uppercase alphabetic and numeric digits are allowed. The first character must not be a digit.

Workstation:

- For file types SEQUENTIAL, INDEXED and RELATIVE, and device type PRINTER, this is the 64-character fully-qualified file specification of the physical file. See the FILE SYSTEM parameter and the *CA-Easytrieve/Workstation User Guide* for more information.
- For file type INDEXED, if the physical data file is defined with an extension, then the indexed file associated with that data file must be explicitly defined on the FILE statement.
- On the workstation, SYSNAME is the 63-character file specification of the physical file name. To maintain portability to the mainframe, use the SYSTEM parameter to specify the physical file name.

UNIX:

- For SEQUENTIAL, INDEXED, and RELATIVE file types and the PRINTER device type, SYSNAME is either a file description string or an environment variable specifying the file description string. If the value of SYSNAME contains a path separator (/), it is treated as a file description string. If it does not contain a path separator, CA-Easytrieve searches for an environment variable with the same name. If found, the value of the variable is used as the file description string. If the variable is not found, the SYSNAME value is used as the path.

  See *File Description String* in the *CA-Easytrieve for UNIX User Guide*.

The length of SYSNAME is limited to the lesser of 256 characters or the maximum path length supported by your UNIX system.

[VIRTUAL] VIRTUAL identifies a file as a CA-Easytrieve Virtual File Manager (VFM) file. CA-Easytrieve virtual files are temporary sequential work files that are normally deleted after the file is read and closed.

[RETAIN] RETAIN inhibits the automatic deletion of a VFM file after it is read. The file is deleted if it is opened for OUTPUT or CREATE in a subsequent JOB activity or at program termination.

```
[MEMORY]
[DISK  ]
```

MEMORY and DISK indicate the type of CICS temporary storage that CA-Easytrieve is to use for storing this file. MEMORY indicates a main storage resident temporary storage queue in CICS. DISK indicates an auxiliary storage resident temporary storage queue. DISK is the default.

[TERMINAL] Code the TERMINAL parameter to route the output for this printer file to an online terminal. The TERMINAL parameter is mutually exclusive with the SPOOL, VIRTUAL, and SYSNAME parameters.

A device type of PRINTER is implied.

```
[   {'terminal-id-literal' } ]
[ID {                      } ]
[   {terminal-id-field-name} ]
```

(CICS only) Specify the name of the destination terminal in *'terminal-id-literal'* or *terminal-id-field-name*. This terminal can be either a display terminal or an online printer. Any valid terminal ID is accepted.

When ID is not specified, output directed to this file is spooled until the file is closed. The output can then be browsed at the originating terminal using the Report Display Facility. You can also then print the file.

[NOFORMFEED] Code NOFORMFEED to indicate that the formfeed character cannot be used to start a new page. If NOFORMFEED is not specified, then CA-Easytrieve can use the formfeed character to start a new page.

```
NEWPAGE [BEFORE]
        [AFTER ]
```

Code NEWPAGE to eject the page each time the file is opened and each time it is closed. Specify NEWPAGE BEFORE to eject the page each time the file is opened. Specify NEWPAGE AFTER to eject the page each time the file is closed.

If NEWPAGE is not specified, CA-Easytrieve does nothing to position the page.

`[SPOOL]` Code the SPOOL parameter to route the output for this printer file to the operating system spooling subsystem. The SPOOL parameter is mutually exclusive with the VIRTUAL, TERMINAL and SYSNAME parameters. In CMS, the printer device must be spooled to RSCS. SPOOL is ignored on the workstation and in UNIX.

A device type of PRINTER is implied.

```
CLASS {'spool-class-literal' }
      {                      }
      {spool-class-field-name}
```

Code CLASS to specify the spool class for the file. CLASS can be specified as a literal or as a field name. Any valid class is accepted.

The default is CLASS 'A'.

```
NODE  {'destination-literal' }
      {                      }
      {destination-field-name}
```

Code NODE to specify the destination for the file. This destination is usually a local or remote printer device name or a network node name, but can be anything meaningful to the operating system spooling subsystem.

NODE can be specified as a literal or as a field name. Any valid node is accepted.

Note: If NODE is not specified, the destination for the file is not passed to the operating system spooling subsystem.

```
       {'user-id-literal' }
USERID {                  }
       {user-id-field-name}
```

Code USERID to specify the user of the printed output. The NODE subparameter must also be specified and must contain a network node name.

USERID can be specified as a literal or as a field name. Any valid userid is accepted.

Note: If the USERID subparameter is not specified, the userid is not passed to the operating system spooling subsystem.

Usage Notes

For compatibility with CA-Easytrieve Plus (batch) programs using VSAM files, both CA-Easytrieve/Online and CA-Easytrieve/Workstation support FILE statements containing VS and its related keywords (shown below).

```
FILE file-name VS ([ES] [F] [PASSWORD 'literal'] +
  CREATE [RESET|UPDATE] [NOVERIFY]
```

See the "File Processing" chapter and the "SQL Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information about using the FILE statement.

Examples

The following examples illustrate FILE statements for various files.

Define a Sequential (SAM) File in MVS:

```
FILE SEQFILE
```

Load an Entry-sequenced, Fixed-length VSAM File:

```
FILE  ENTSEQ SEQUENTIAL F CREATE RESET
```

Define a Virtual File with RETAIN:

```
FILE VRTFILE V(200)    +
     VIRTUAL RETAIN
```

Define a Printer File to be Viewed at the Terminal:

```
FILE PRTFILE PRINTER (PAGESIZE 20 LINESIZE 80)   +
     TERMINAL
```

Chapter 5

Statements G - M

GET Statement

GET places the next or previous sequential record of the named file into the file's record buffer.

Syntax

```
                      [HOLD  ]
GET file-name [PRIOR] [      ] [STATUS]
                      [OHOLD ]
```

Parameters

file-name *File-name* identifies the input file defined in the library section. *File-name* can be any file type except SQL.

[PRIOR] Specify PRIOR to place the previous sequential record of the named file into the file's record buffer. If you specify PRIOR and the position in the file is not established, the last record in the file is placed in the file's record buffer.

Note: The access method of the operating system must support retrieval of previous records or an execution error occurs.

```
[HOLD  ]
[NOHOLD]
```

Except in CICS, CA-Easytrieve automatically issues a hold request for records when UPDATE is specified on the FILE statement. You use NOHOLD to override this process. In CICS, NOHOLD is the default.

Specify HOLD to hold a record for update. HOLD is invalid if UPDATE is not specified on the FILE statement. HOLD does not mean you are required to perform the update. It holds the position in the file and can also lock the record (workstation LANs). Records are automatically released when the update operation completes or a commit point is taken. You can also manually release the hold on any record with the RELEASE statement.

NOHOLD specifies that a record is not held for update.

Note: You cannot browse (GET) a file in CICS if you specify HOLD. An execution error occurs. When you want to update a record, use the READ statement.

`[STATUS]` Specify STATUS whenever the possibility exists for an unsatisfactory completion of the input/output request.

STATUS checks input/output processing to see if it was performed properly. STATUS causes the file's FILE-STATUS field to be set with the appropriate return code. See Appendix A, "System-Defined Fields," to determine the meaning of the contents of FILE-STATUS. Normally, a zero or non-zero test is sufficient.

Note: FILE-STATUS is not defined if you do not specify a file type parameter on the FILE statement.

If you do not code STATUS and the operating system returns a non-zero status, CA-Easytrieve issues an appropriate diagnostic message.

Usage Notes

You must test for end-of-file (EOF) or file presence (IF *file-name*) when using the GET statement to ensure record availability. If you specified GET PRIOR, an EOF means you have reached the beginning of the file.

When you reverse the direction of a GET statement by using GET PRIOR, the record returned is the record immediately preceding the record previously placed in the file's record buffer. When you reverse the direction of a GET PRIOR statement by using only GET, the record returned is the record immediately following the record previously placed in the field's record buffer.

You cannot issue a GET PRIOR statement following a POINT statement, or a GET statement following a POINT PRIOR statement. See the POINT Statement for more information.

You cannot use GET for a file designated as automatic input. To inhibit automatic input, specify INPUT NULL on the JOB statement:

```
JOB INPUT NULL
```

You can use GET to access a secondary file while automatically accessing a primary file.

Examples

The following code illustrates the use of the GET statement.

```
FILE PERSNL INDEXED
```

```
%PERSNL
PROGRAM NAME MYPROG
   GET PERSNL STATUS
   IF PERSNL:FILE-STATUS NE 0
     DISPLAY PERSNL:FILE-STATUS
   ELSE
     DISPLAY HEX PERSNL
   END-IF
```

The following code illustrates testing for EOF when using the GET statement.

```
FILE MASTER
...
GET MASTER
  IF EOF MASTER
     STOP
  END-IF
...
```

GOTO Statement

The GOTO statement allows you to modify the natural top to bottom logic flow of statement execution.

Syntax

```
{GOTO }  {label }
{     }  {JOB   }
{GO TO}  {SCREEN}
```

Parameters

{*label*} Specify *label* to immediately transfer execution control to the first statement following the associated *label*. Processing then continues in a top-to-bottom sequence. The *label* must be contained in the same activity or procedure.

A *label* can:

- Be up to 128 alphanumeric characters in length
- Contain any character other than a delimiter
- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

A statement *label* is a complete CA-Easytrieve statement that you can code before the following statements:

Assignment	CALL
CASE	CLOSE
COMMIT	CURSOR

- DELETE
- DISPLAY
- DLI
- DO
- END-DO
- END-IF
- END-PROC
- EXECUTE
- EXIT
- FETCH
- GET
- GOTO
- IDMS
- IF
- INSERT
- MESSAGE
- MOVE
- MOVE LIKE
- PERFORM
- POINT
- PRINT
- PUT
- READ
- REFRESH
- RELEASE
- RESHOW
- ROLLBACK
- SEARCH
- SELECT (except non-file SQL)
- SET
- SQL
- Statement label
- STOP
- TRANSFER
- UPDATE
- WRITE

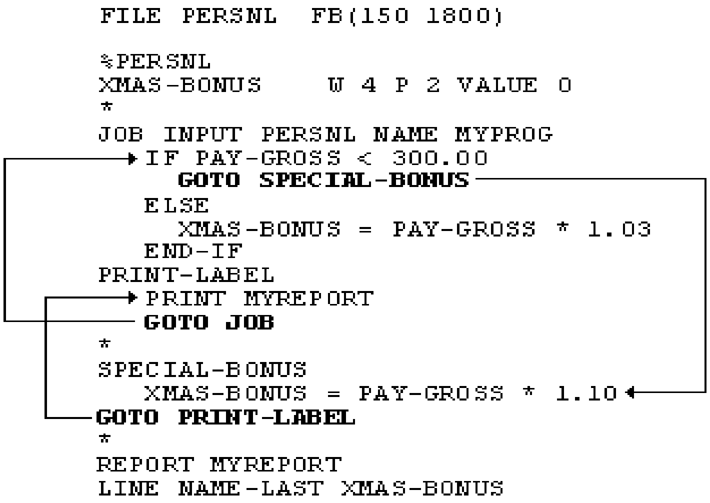
`{JOB}` GOTO JOB causes an immediate branch to the top of the current JOB activity. It does not include execution of the START procedure. When used in a START procedure, GOTO JOB terminates the START procedure. When used in a FINISH procedure, GOTO JOB terminates the FINISH procedure.

`{SCREEN}` GOTO SCREEN branches immediately to the top of the current SCREEN activity, including execution of the BEFORE-SCREEN procedure. It does not include execution of the INITIATION procedure. When used in an INITIATION procedure, GOTO SCREEN terminates the INITIATION procedure.

GOTO SCREEN cannot be coded in a BEFORE-SCREEN procedure. If GOTO SCREEN is coded in a TERMINATION procedure, GOTO SCREEN terminates the screen activity.

Example

The following example illustrates the use of GOTO in a program. The arrows indicate that control is passed to the first executable statement following the label or job statement.



```
FILE  PERSNL   FB(150 1800)

%PERSNL
XMAS-BONUS     W 4 P 2 VALUE 0
*
JOB  INPUT  PERSNL  NAME  MYPROG
   IF PAY-GROSS < 300.00
     GOTO  SPECIAL-BONUS
   ELSE
     XMAS-BONUS  =  PAY-GROSS  *  1.03
   END-IF
PRINT-LABEL
   PRINT  MYREPORT
   GOTO JOB
*
SPECIAL-BONUS
   XMAS-BONUS  =  PAY-GROSS  *  1.10
GOTO  PRINT-LABEL
*
REPORT MYREPORT
LINE  NAME-LAST  XMAS-BONUS
```

GRAPH Statement

The GRAPH statement defines the style and characteristics of a graph.

Note: Graph processing is available only when using CA-Easytrieve/Workstation.

Syntax

```
                              [      {'LOW'           }]
GRAPH [graph-name] [SUMMARY] [MODE {'HIGH'          }]  +
                              [      {mode-field-name}]

     [      {'PIE'           }]
     [      {'VBAR'          }]
     [      {'SVBAR'         }]
     [      {'HBAR'          }]
     [STYLE {'SHBAR'         }] [NOHEADING]
     [      {'LINE'          }]
     [      {'XY'            }]
     [      {'SCATTER'       }]
     [      {style-field-name}]
```

Parameters

[*graph-name*] Specify the name of the graph. *Graph-name* is optional when there is only one graph coded in a JOB activity. If you code multiple graphs, only the first graph can be unnamed. Each *graph-name* must be unique in the JOB activity.

At least one *graph-name* must be coded for each DRAW *graph-name* statement. For unnamed graphs, you can code the DRAW statement without the *graph-name* parameter.

[SUMMARY] Specify SUMMARY to automatically sum all values (y-value) for each category (x-value) before the graph is displayed.

```
[    {'LOW'          }]
[MODE{'HIGH'         }]
[    {mode-field-name}]
```

MODE determines the display resolution of the graph.

'LOW' specifies a resolution of 640 x 200 in black and white. 'LOW' resolution mode is compatible with the CGA, EGA, VGA and MCGA video adapter boards. 'LOW' is recommended if you are going to print the graph.

'HIGH' specifies a resolution of 640 x 350 in 16 colors and is compatible with the EGA and VGA video adapter boards.

'Mode-field-name' is a field you can define that contains 'LOW' or 'HIGH.'

If not specified, MODE defaults to the highest resolution supported by the video adapter board.

```
[      {'PIE'           }]
[      {'VBAR'          }]
[      {'SVBAR'         }]
[      {'HBAR'          }]
[STYLE {'SHBAR'         }]
[      {'LINE'          }]
[      {'XY'            }]
[      {'SCATTER'       }]
[      {style-field-name}]
```

STYLE specifies the style of graph to be displayed.

Specify 'PIE' to display a pie chart. 'PIE' is the default if the graph style is not coded. The y-value for this graph determines the size of the pie slice. The x-value for this graph determines the category for the y-value. If you code SUMMARY, all y-values for each identical x-value are summed producing a slice that is the size of the sum of all of the y-values for this category.

Specify 'VBAR' or 'SVBAR' to display a vertical bar graph. The y-value determines the height of the vertical bar. The x-value determines the category for the y-value. If you code SUMMARY, all y-values for each identical x-value are summed, producing a vertical bar that is the sum of all of the y-values for this category.

‘VBAR’ produces side-by-side vertical bars if multiple y-values are coded on the VALUE statement. ‘SVBAR’ produces stacked vertical bars if multiple y-values are coded on the VALUE statement.

Specify ‘HBAR’ or ‘SHBAR’ to display a horizontal bar graph. The y-value determines the length of the horizontal bar. The x-value determines the category for the y-value. If you code SUMMARY, all y-values for each identical x-value are summed producing a horizontal bar that is the sum of all of the y-values for this category.

‘HBAR’ produces side-by-side horizontal bars if multiple y-values are coded on the VALUE statement. ‘SHBAR’ produces stacked horizontal bars if multiple y-values are coded on the VALUE statement.

Specify ‘LINE’ to display a line graph. The y-value determines the height of the data point on the graph. The x-value determines the category for the y-value. If you code SUMMARY, all y-values for each identical x-value are summed producing a data point that is the sum of all of the y-values for this category.

Specify ‘XY’ to display an XY graph in which values are connected by lines. The y-value determines position of the data point on the y-axis. The x-value determines the position of the data point on the x-axis. If you code SUMMARY, all y-values for each identical x-value are summed producing a data point that is the sum of all of the y-values for this x-value.

Specify ‘SCATTER’ to display a scatter graph. Values are not connected by lines and a y-axis grid is displayed. The y-value determines position of the data point on the y-axis. The x-value determines the position of the data point on the x-axis. If you code SUMMARY, all y-values for each identical x-value are summed producing a data point that is the sum of all of the y-values for this x-value..

‘Style-field-name’ is a field you can define that contains the graph type (one of the style literals described above).

See the “Graph Processing” chapter in the *CA-Easytrieve Programmer Guide*, for an example of each type of graph.

`[NOHEADING]` Specify NOHEADING to inhibit the display of X and Y headings. The default is for X and Y field headings to be displayed on the graph. Headings are ignored for PIE graphs.

Example

The following program code produces a vertical bar graph in low resolution mode that displays the sum of the gross pay for each region without axis headings.

```
FILE PERSNL F(150)
%PERSNL
JOB INPUT PERSNL NAME DRAW-GRAPH
  DRAW GRAPH1
GRAPH GRAPH1 STYLE 'VBAR' MODE 'LOW' SUMMARY NOHEADING
  TITLE 'GROSS PAY BY REGION'
  VALUE REGION PAY-GROSS
```

HEADING Statement

The HEADING statement optionally defines an alternate heading for a field. When defining a field, you specify the default heading. Using the HEADING statement in a report or on a graph allows you to override the default field headings for that report or graph.

Syntax

```
HEADING field-name ([#font-number] 'heading-literal'...)
```

Parameters

field-name For reports, *field-name* specifies a field in your program. The heading you define is used for fields identified on LINE 01 of your report declaration.

For graphs, *field-name* specifies a field defined on the VALUE statement of your graph declaration.

[*#font-number*] *(Mainframe and UNIX only)* #*Font-number* defines the number of a font that CA-Easytrieve uses to format *'heading-literal'* in the heading area of a report. You can only specify #*font-number* if you direct the report to an extended reporting printer. If you direct the report to a normal printer, a syntax error occurs when you code #*font-number*. You can specify a unique font index for each *'heading-literal'* by coding the # sign and a value for #*font-number* before *'heading-literal'*. Any *'heading-literal'* that does not have a font index assigned uses the default font for the assigned extended reporting printer.

('*heading-literal*'...) *'Heading-literal'* can be up to 128 characters in length.

For reports, a single line of alphanumeric text replaces the default header and prints as a header over a column or field. Multiple literals, each enclosed within single quotes (') and separated by one or more blanks within the parentheses, are stacked vertically over the column or field when printed.

For graphs, specify the text for the heading to be displayed for the field on the VALUE statement. Multiple literals, each enclosed within single quotes (') and separated by one or more blanks within the parentheses, are displayed on a single line with a space between each literal.

Usage Notes

The HEADING statement overrides default field headings defined in the library section. The HEADING statement also provides alternate heading capabilities for system-defined fields such as TALLY and LEVEL.

The HEADING statement is ignored for PIE graphs.

Examples

The following example illustrates various report heading options:

```
Statements:

FILE PERSNL FB(150 1800)
  SSN             4   5 P MASK '999-99-9999' +
        HEADING('SOCIAL' 'SECURITY' 'NUMBER')
  EMPNAME           17   20 A
    NAME-LAST  EMPNAME     8 A
    NAME-FIRST EMPNAME +8 12 A
  PAY-NET         90    4 P 2
JOB INPUT PERSNL NAME MYPROG
  PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65
  HEADING PAY-NET ('NET', 'PAY')
  LINE EMPNAME SSN '* NO OVERTIME *' PAY-NET

Results:

                        SOCIAL
                       SECURITY                          NET
       EMPNAME          NUMBER                           PAY


   WIMN    GLORIA     025-30-5228   * NO OVERTIME *     251.65
   BERG    NANCY      121-16-6413   * NO OVERTIME *     547.88
```

The next example produces a vertical bar graph that displays the sum of the gross pay for each region with user-specified headings:

```
FILE PERSNL F(150)
%PERSNL
JOB INPUT PERSNL NAME DRAW-GRAPH
  DRAW GRAPH1
```

```
GRAPH VBAR GRAPH1 SUMMARY MODE('HIGH') TYPE('VBAR')
  SEQUENCE REGION
  HEADING REGION ('Region')
  HEADING PAY-GROSS ('Gross' 'Pay')
  TITLE COL 1 SYSDATE 'GROSS PAY BY REGION' COL 73 SYSTIME
  VALUE REGION PAY-GROSS
```

IDD FILE Statement

The IDD FILE statement identifies a non-CA-IDMS file in the IDD and builds file and field definitions.

Syntax

```
                   [        {HIGHEST}]
IDD FILE file-name [VERSION {LOWEST }]  +
                   [        {nnnn   }]

                   [SELECT (record-name...)]  +

                   [FILENAME new-file-name]
```

Parameters

`file-name` *File-name* is the one to 32-character name that specifies the IDD file. *File-name* becomes the name of the FILE created by the IDD FILE statement. To override this name, see the FILENAME parameter.

```
[        {HIGHEST}]
[VERSION {LOWEST }]
[        {nnnn   }]
```

Specify HIGHEST, LOWEST, or *nnnn* (a positive integer) as the version of the file.

`[SELECT (record-name ...)]` *Record-name* is a one to 32-character name that identifies a record of the *file-name* defined. Repeat the *record-name* to identify as many records as needed. Omit the SELECT clause if you want to define all the field definitions for the *file-name*.

`[FILENAME new-file-name]` *New-file-name* is a one to 128-character name specifying the name of the file created by the IDD FILE statement.

Usage Notes

The *file-name* can be qualified by the *file-name*'s version. All records defined within the file are used to generate the file's field definitions unless the optional SELECT parameter identifies specific records to be used.

IDD NAME Statement

The IDD NAME statement establishes or re-establishes the dictionary entity retrieval environment. The IDD NAME statement specifies the program name, the database name of the data dictionary, the Central Version Node, and the Secondary Load Area's dictionary name and dictionary node.

Syntax

```
IDD NAME [PROGRAM-NAME 'program-literal']  +

         [DBNAME 'db-name-table-literal']  +

         [NODE 'node-literal']  +

         [DICTNAME 'dictionary-literal']  +

         [DICTNODE 'dictionary-node-literal']
```

Parameters

[PROGRAM-NAME '*program-literal*'] *'Program-literal'* identifies the Program Name used to access an authorized subschema. *'Program-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

[DBNAME '*db-name-table-literal*'] *'Db-name-table-literal'* identifies the DB Name Table of the data dictionary that contains definitions of schema, subschema, records, and fields. *'Db-name-table-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

[NODE '*node-literal*'] *'Node-literal'* specifies the CA-IDMS Central Version NODE that will process CA-Easytrieve IDD requests. *'Node-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

[DICTNAME '*dictionary-literal*'] *'Dictionary-literal'* identifies the Secondary Load Area dictionary name. *'Dictionary-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

[DICTNODE '*dictionary-node-literal*'] *'Dictionary-node-literal'* identifies the Secondary Load Area dictionary node. *'Dictionary-node-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

Usage Notes

IDD entities are retrieved from the designated dictionary/node until the environment is altered by issuing another IDD NAME statement. You can use the IDD NAME statement as many times as required and before any other IDD statement. However, the PROGRAM-NAME parameter can be used only once.

IDD RECORD Statement

The IDD RECORD statement identifies and defines CA-IDMS and non-CA-IDMS records. The *record-name* can be qualified by the record's version. The elements defined within the record are used to generate field definitions at the location specified.

Syntax

```
                         [        {HIGHEST}]
IDD RECORD record-name [VERSION {LOWEST }]  +
                         [        {nnnn   }]

  [         {start-position}]
  [         {* [+offset]    }]
  [LOCATION {                }]
  [         {W               }]
  [         {S               }]
```

Parameters

record-name The *record-name* is the one to 32-character name that specifies the IDD logical record or the IDD standard record.

```
[        {HIGHEST}]
[VERSION {LOWEST }]
[        {nnnn   }]
```

Specify HIGHEST, LOWEST, or *nnnn* (a positive integer) as the version of the file.

[LOCATION] Use this optional parameter to specify the location at which the field definitions will be generated. If you do not specify LOCATION, W (a W-type working storage field) is the default.

{*start-position*} *Start-position* specifies the starting position relative to position one of the record.

{* [+*offset*]} The * (asterisk) indicates that the field begins in the next available starting position (highest position assigned so far, plus 1). The optional +*offset* is an offset you want added to the * value. There must be at least one blank between the * and the optional +*offset*.

{W *or* S} Coding W or S establishes a working storage field. W fields are spooled to report (work) files; S fields are not. W is the default location if the LOCATION parameter is not coded.

IDD SUBSCHEMA Statement

The IDD SUBSCHEMA statement identifies the subschema and builds the file, record, logical record, element record, and field definitions for a subschema. The subschema can be qualified by the schema and schema's version.

Syntax

```
IDD SUBSCHEMA subschema-name  +

                 [                 [        {HIGHEST}]]
                 [SCHEMA schema-name [VERSION {LOWEST }]] +
                 [                 [        {nnnn   }]]

                 [RESET]   +

                 [SELECT (record-name...)]  +

                 [FILENAME file-name]
```

Parameters

subschema-name *Subschema-name* is a one to eight-character name specifying the SUBSCHEMA that contains the record and field definitions to be retrieved. *Subschema-name* becomes the name of the FILE created by the IDD SUBSCHEMA statement. To override this name, see the FILENAME parameter.

```
[                  [        {HIGHEST}]]
[SCHEMA schema-name [VERSION {LOWEST }]]
[                  [        {nnnn   }]]
```

Schema-name is the one to eight-character name that specifies the schema that owns the subschema (when a subschema can be owned by multiple schemas). The optional VERSION parameter specifies the version of the schema.

[RESET] The optional RESET parameter requests that all element records under control of RETRIEVE be reset to binary zero immediately prior to retrieving each root record.

[SELECT (*record-name* ...)] The optional SELECT clause identifies specific subschema records. *Record-name* is a one to 32-character name that identifies a record for which the definition is accessed. Repeat *record-name* to specifically identify all the records you need. You can omit the SELECT parameter to access all the database records (but not logical records) defined for the subschema.

`[FILENAME` *`file-name`*`]` *File-name* is a one to 128-character name specifying the name of the file created by the IDD SUBSCHEMA statement.

Usage Notes

You can use the optional SELECT parameter to request that only specific records be defined. If the SELECT parameter is omitted, the definitions of logical records are not generated. Only database records can be defined in this way. The SELECT parameter must be used to generate logical record definitions.

IDD VERSION Statement

Use the IDD VERSION statement to set a global override of the Site Options Table VERFILE, VERREC, and VERSCHM defaults.

Syntax

```
             [       {HIGHEST}]
IDD VERSION [SCHEMA {LOWEST }]  +
             [       {nnnn   }]

             [     {HIGHEST}]
             [FILE {LOWEST }]  +
             [     {nnnn   }]

             [       {HIGHEST}]
             [RECORD {LOWEST }]
             [       {nnnn   }]
```

Parameters

```
[       {HIGHEST}]
[SCHEMA {LOWEST }]
[       {nnnn   }]
```

Specify HIGHEST, LOWEST, or a positive integer (*nnnn*) to identify the default version of the SCHEMA. Any request to retrieve a subschema that specifies a schema but not a version uses this value.

```
[     {HIGHEST}]
[FILE {LOWEST }]
[     {nnnn   }]
```

Specify HIGHEST, LOWEST, or a positive integer (*nnnn*) to identify the default version of any FILE to be retrieved when the version is not specified on the IDD FILE statement.

```
[       {HIGHEST}]
[RECORD {LOWEST }]
[       {nnnn   }]
```

Specify HIGHEST, LOWEST, or a positive integer (*nnnn*) to identify the default version of any RECORD to be retrieved when the version is not specified on the IDD RECORD statement.

Usage Notes

Other IDD statements appearing after the IDD VERSION statement, that have VERSION parameters as part of their syntax but no VERSION parameter coded, default to the version specified in the IDD VERSION statement. IDD statements that have a VERSION parameter coded override the VERSION statement. The defaults specified by the IDD VERSION statement remain in effect until another IDD VERSION statement is issued.

You can code as many IDD VERSION statements as you require. If the IDD VERSION statement is not used, the default versions are retrieved from the Site Options Table.

IDMS ACCEPT DBKEY Statement

The IDMS ACCEPT DBKEY statement transfers database keys to program storage. The IDMS ACCEPT DBKEY statement has two formats. Format 1 returns the current database key for the record, set, or area specified. Format 2 returns the database key that is the next, prior, or owner of the specified set.

Syntax

Format 1

```
                                    [{RECORD} {currency-field-name}]
      IDMS ACCEPT DBKEY receive-field-name [{AREA  } {                   }]
                                    [{SET   } {'currency-literal' }]
```

Format 2

```
                                    {NEXT } {set-field-name}
      IDMS ACCEPT DBKEY receive-field-name {PRIOR} {              }
                                    {OWNER} {'set-literal' }
```

Parameters

Format 1

receive-field-name *Receive-field-name* identifies the four-byte binary field to receive the specified database key.

```
{RECORD}
{AREA  }
{SET   }
```

Specify RECORD, SET, or AREA.

```
{currency-field-name}
{                   }
{'currency-literal' }
```

Currency-field-name or *'currency-literal'* identifies the currency for the desired key. *Currency-field-name* must be a 16-byte alphanumeric field. *'Currency-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value. The default is the current record of the run-unit.

Format 2

`receive-field-name` *Receive-field-name* identifies the four-byte binary field to receive the specified database key.

```
{NEXT }
{PRIOR}
□OWNER}
```

Specify NEXT, PRIOR, or OWNER.

```
{set-field-name}
{              }
{'set-literal' }
```

Set-field-name or *'set-literal'* identifies the name of the desired set. *Set-field-name* must be a 16-byte alphanumeric field. *'Set-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

IDMS ACCEPT PAGE-INFO Statement

The IDMS ACCEPT PAGE-INFO statement transfers database page information to program storage.

Syntax

```
                                        {      } {currency-field-name}
IDMS ACCEPT PAGE-INFO receive-field-name {RECORD} {                   }
                                        {      } {'currency-literal' }
```

Parameters

`receive-field-name` *Receive-field-name* identifies the four-byte binary field to receive the page information.

```
{currency-field-name}
{                   }
{'currency-literal' }
```

Currency-field-name or *'currency-literal'* identifies the record name for the desired key. *Currency-field-name* must be a 16-byte alphanumeric field. *'Currency-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

IDMS ACCEPT PROCEDURE Statement

The IDMS ACCEPT PROCEDURE statement returns information from the Application Program Information Block (APIB) associated with a database procedure to the program.

Syntax

```
                       {proc-field-name}
IDMS ACCEPT PROCEDURE {                } TO apib-field-name
                       {'proc-literal' }
```

Parameters

```
{proc-field-name}
{               }
{'proc-literal' }
```

Proc-field-name or *'proc-literal'* identifies the name of a DBA-written database procedure. *Proc-field-name* must be an eight-byte alphanumeric field. *'Proc-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

TO *apib-field-name* *Apib-field-name* is a 256-byte alphanumeric area of storage to which the APIB is copied.

IDMS ACCEPT STATISTICS Statement

The IDMS ACCEPT STATISTICS statement retrieves the system statistics.

Syntax

```
IDMS ACCEPT STATISTICS stat-field-name
```

Parameters

stat-field-name *Stat-field-name* identifies a 100-byte field that you must define in working storage to receive the current system runtime statistical information.

Usage Notes

See your CA-IDMS documentation for more information about the statistics produced.

IDMS BIND Statement

The IDMS BIND statement signs on the activity with the database management system.

Syntax

```
          {subschema-name     }
IDMS BIND {                   }  +
          {'subschema-literal'}

       [             {program-name     }]
       [PROGRAM-NAME {                 }]  +
       [             {'program-literal'}]

       [       {db-name-table-name     }]
       [DBNAME {                       }]  +
       [       {'db-name-table-literal'}]

       [     {node-name     }]
       [NODE {              }]  +
       [     {'node-literal'}]

       [         {dictionary-name     }]
       [DICTNAME {                    }]  +
       [         {'dictionary-literal'}]

       [         {dictionary-node-name     }]
       [DICTNODE {                         }]
       [         {'dictionary-node-literal'}]
```

Parameters

```
{subschema-name     }
{                   }
{'subschema-literal'}
```

Subschema-name identifies the subschema to be processed with CA-IDMS. *Subschema-name* must be an eight-byte alphanumeric field. *'Subschema-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[             {program-name     }]
[PROGRAM-NAME {                 }]
[             {'program-literal'}]
```

Program-name or *'program-literal'* specifies the name used to identify the program to CA-IDMS during execution. *Program-name* must be an eight-byte alphanumeric field. *'Program-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[       {db-name-table-name     }]
[DBNAME {                       }]
[       {'db-name-table-literal'}]
```

Db-name-table-name or *'db-name-table-literal'* specifies a DB Name Table. Data retrieved during execution of the user's program will be from the named database. *Db-name-table-name* must be an eight-byte alphanumeric field. *'Db-name-table-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[     {node-name     }]
[NODE {              }]
[     {'node-literal'}]
```

Node-name or *'node-literal'* specifies the Central Version Node that will host the CA-IDMS activity generated by the user's program. *Node-name* must be an eight-byte alphanumeric field. *'Node-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[         {dictionary-name     }]
[DICTNAME {                    }]
[         {'dictionary-literal'}]
```

Dictionary-name or *'dictionary-literal'* specifies the Dictionary Name of a Secondary Load Area. *Dictionary-name* must be an eight-byte alphanumeric field. *'Dictionary-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[         {dictionary-node-name     }]
[DICTNODE {                         }]
[         {'dictionary-node-literal'}]
```

Dictionary-node-name or *'dictionary-node-literal'* specifies the Dictionary Node of a Secondary Load Area. *Dictionary-node-name* must be an eight-byte alphanumeric field. *'Dictionary-node-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

IDMS BIND FILE Statement

The IDMS BIND FILE statement gives the database management system access to the record in program storage.

Syntax

```
IDMS BIND FILE file-name RECORD record-name
```

Parameters

file-name *File-name* identifies the file where the record-area is to be allocated.

RECORD *record-name* *Record-name* identifies the record to be bound with CA-IDMS.

IDMS BIND PROCEDURE Statement

The IDMS BIND PROCEDURE statement establishes communications between a program and a DBA-written database procedure.

Syntax

```
                        {proc-field-name}
IDMS BIND PROCEDURE {                } TO receive-field-name
                        {'proc-literal' }
```

Parameters

```
{proc-field-name}
{               }
{'proc-literal' }
```

Proc-field-name or *'proc-literal'* identifies the name of a DBA-written database procedure. *Proc-field-name* must be an eight-byte alphanumeric field. *'Proc-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

TO *receive-field-name* *Receive-field-name* must be a 256-byte alphanumeric field. The contents of *receive-field-name* are copied to the APIB as part of the execution of the IDMS BIND PROCEDURE statement.

IDMS COMMIT Statement

The IDMS COMMIT statement requests the creation of a checkpoint.

Syntax

```
IDMS COMMIT [ALL]
```

Parameters

`[ALL]` This optional parameter controls which locks are released. The default is ALL EXCEPT THOSE HELD.

IDMS CONNECT Statement

The IDMS CONNECT statement establishes a record as a member of a set occurrence.

Syntax

```
                    {record-field-name}     {set-field-name}
IDMS CONNECT RECORD {                 } SET {              }
                    {'record-literal' }     {'set-literal' }
```

Parameters

```
{record-field-name}
{                 }
{'record-literal' }
```

Record-field-name or *'record-literal'* identifies the record to be connected. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
{set-field-name}
{              }
{'set-literal' }
```

Set-field-name or *'set-literal'* specifies the set to which the record is to be connected. *Set-field-name* must be a 16-byte alphanumeric field. *'Set-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

IDMS DISCONNECT Statement

The IDMS DISCONNECT statement cancels the relationship between a record and a set occurrence.

Syntax

```
                         {record-field-name}     {set-field-name}
IDMS DISCONNECT RECORD {                   } SET {               }
                         {'record-literal' }     {'set-literal' }
```

Parameters

```
       {record-field-name}
RECORD {                 }
       {'record-literal' }
```

Record-field-name or *'record-literal'* identifies the record to be disconnected. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
    {set-field-name}
SET {              }
    {'set-literal' }
```

Set-field-name or *'set-literal'* specifies the set from which the record is to be disconnected. *Set-field-name* must be a 16-byte alphanumeric field. *'Set-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

IDMS ERASE Statement

Format 1 of the IDMS ERASE statement makes a record unavailable for further processing and removes it from all set occurrences in which it participates as a member. Format 2 makes a logical record unavailable for further processing.

Syntax

Format 1

```
                                      [MEMBERS  ]
                  {record-field-name} [PERMANENT]
IDMS ERASE RECORD {                 } [         ]
                  {'record-literal' } [SELECTIVE]
                                      [ALL      ]
```

Format 2

```
IDMS ERASE RECORD logical-record-name [WHERE (boolean-expression)]
```

Parameters

Format 1

```
       {record-field-name}
RECORD {                 }
       {'record-literal' }
```

Record-field-name or *'record-literal'* identifies the record to be erased. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
[MEMBERS  ]
[PERMANENT]
[         ]
[SELECTIVE]
[ALL      ]
```

This optional parameter controls the type of erasure. The default is MEMBERS.

Format 2

RECORD *logical-record-name* *Logical-record-name* is a one to sixteen-character name that identifies the logical record to be erased. *Logical-record-name* must be the name of a logical record defined by a LOGICAL-RECORD statement.

[WHERE (*boolean-expression*)] Code the optional WHERE clause to provide a Boolean expression CA-IDMS uses to select the logical records to be erased.

IDMS FIND/OBTAIN Statement

The IDMS FIND and IDMS OBTAIN statements are described together because their formats are the same. The IDMS FIND statement only locates (positions to) a record; the IDMS OBTAIN statement locates and then retrieves a record. These statements have six formats:

- Format 1 locates/retrieves a record based on its DBKEY
- Format 2 locates/retrieves the current occurrence of the record type, set, or area
- Format 3 locates/retrieves a record within a set or area
- Format 4 locates/retrieves the owner record within the set
- Format 5 locates/retrieves a record based on its CALC key
- Format 6 locates retrieves an ordered (sorted) record from a set.

Note: To retrieve logical records, see the IDMS OBTAIN Statement, described separately later in this chapter.

Syntax

Format 1

```
                                       [            {record-field-name}]
                                       [   RECORD   {                  }]
     {FIND  }       {key-field-name}   [            {'record-literal'  }]  [SHARE|SHR     ]
IDMS {      } DBKEY {               }  [                                ]  [              ]
     {OBTAIN}       {'key-literal'  }  [            {page-field-name   }]  [EXCLUSIVE|EXC]
                                       [  PAGE-INFO {                  }]
                                       [            {'page-literal'    }]
```

Format 2

```
     {FIND  }          [{RECORD} {field-name}]  [SHARE|SHR     ]
IDMS {      } CURRENT  [{SET   } {          }]  [              ]
     {OBTAIN}          [{AREA  } {'literal' }]  [EXCLUSIVE|EXC]
```

Format 3

```
     {FIND  }▯ {NEXT                       }   [        {record-field-name}]
IDMS {      }▯ {PRIOR                      }   [RECORD  {                 }]  +
     {OBTAIN}▯ {FIRST                      }   [        {'record-literal' }]
              {LAST                        }
              {      {nth-field-name}      }
              {NTH   {              }      }
              {      {nth-literal   }      }

     {SET } {search-field-name} [SHARE|SHR     ]
     {    } {                 } [              ]
     {AREA} {'search-literal' } [EXCLUSIVE|EXC]
```

Format 4

```
     {FIND  }            {set-field-name} [SHARE|SHR     ]
IDMS {      } OWNER SET  {              } [              ]
     {OBTAIN}            {'set-literal' } [EXCLUSIVE|EXC]
```

Format 5

```
     {FIND  } {CALC     }         {record-field-name} [SHARE|SHR     ]
IDMS {      } {         } RECORD  {                 } [              ]
     {OBTAIN} {DUPLICATE}         {'record-literal' } [EXCLUSIVE|EXC]
```

Format 6

```
     {FIND  }                   {record-field-name}     {set-field-name}
IDMS {      } [CURRENT] RECORD  {                 } SET {              }  +
     {OBTAIN}                   {'record-literal' }     {'set-literal' }

     {control-field-name}        [SHARE|SHR     ]
     USING ({                  }...)  [              ]
     {'control-literal' }        [EXCLUSIVE|EXC]
```

Parameters

Format 1

```
      {key-field-name}
DBKEY {              }
      {'key-literal' }
```

Key-field-name or *'key-literal'* identifies the key of the database record. *Key-field-name* must be a four-byte binary field. *'Key-literal'* must be a four-byte hexadecimal value.

```
[       {record-field-name}]
[RECORD{                 }]
[       {'record-literal' }]
```

Record-field-name or *'record-literal'* identifies the record to be located/retrieved. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value. The default is any record type that satisfies the DBKEY.

```
[           {page-field-name}]
[PAGE-INFO {                }]
[           {'page-literal' }]
```

Page-field-name or *'page-literal'* identifies the record to be located/retrieved by its page information. *Page-field-name* must be a 4-byte binary field. *'Page-literal'* must be a 4-byte hexadecimal value.

```
[SHARE|SHR    ]
[             ]
[EXCLUSIVE|EXC]
```

These optional parameters determine the type of lock to be placed on the object record.

Format 2

```
[{RECORD} {field-name}]
[{SET   } {          }]
[{AREA  } {'literal' }]
```

Field-name or *'literal'* identifies the record, set, or area to be located/retrieved. *Field-name* must be a 16-byte alphanumeric field. *'Literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value. The default locates/retrieves the current record of the run-unit.

```
[SHARE|SHR    ]
[             ]
[EXCLUSIVE|EXC]
```

These optional parameters determine the type of lock to be placed on the object record.

Format 3

```
{NEXT                    }
{PRIOR                   }
{FIRST                   }
{LAST                    }
{       {nth-field-name} }
{NTH    {              } }
{       {nth-literal   } }
```

Nth-field-name or *nth-literal* identifies the record occurrence of the set or area to be located/retrieved. *Nth-field-name* must be a four-byte binary integer. *Nth-literal* must be a positive or negative integer.

```
[        {record-field-name}]
[RECORD {                 }]
[        {'record-literal' }]
```

Record-field-name or *'record-literal'* identifies the record to be located/retrieved. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value. The default is the record that otherwise satisfies the search criteria.

```
{SET }{search-field-name}
{    }{                 }
{AREA}{'search-literal' }
```

Search-field-name or *'search-literal'* identifies the set or area that determines the scope of the search. *Search-field-name* must be a 16-byte alphanumeric field. *'Search-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
[SHARE|SHR    ]
[             ]
[EXCLUSIVE|EXC]
```

These optional parameters determine the type of lock to be placed on the object record.

Format 4

```
          {set-field-name}
OWNER SET {              }
          {'set-literal' }
```

Set-field-name or *'set-literal'* identifies the set to search. *Set-field-name* must be a 16-byte alphanumeric field. *'Set-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
[SHARE|SHR    ]
[             ]
[EXCLUSIVE|EXC]
```

These optional parameters determine the type of lock to be placed on the object record.

```
{CALC     }
{         }
{DUPLICATE}
```

This parameter determines whether the first (CALC) or next (DUPLICATE) record is located/retrieved.

```
       {record-field-name}
RECORD {                 }
       {'record-literal' }
```

Record-field-name or *'record-literal'* identifies the record to locate/retrieve. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
[SHARE|SHR    ]
[             ]
[EXCLUSIVE|EXC]
```

These optional parameters determine the type of lock to be placed on the object record.

Format 6

[CURRENT] This optional parameter controls the start of the search. The default begins with the owner of the current record within the set.

```
       {record-field-name}
RECORD {                 }
       {'record-literal' }
```

Record-field-name or *'record-literal'* identifies the record to locate/retrieve. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
    {set-field-name}
SET {              }
    {'set-literal' }
```

Set-field-name or *'set-literal'* identifies the set to search. *Set-field-name* must be a 16-byte alphanumeric field. *'Set-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
       {control-field-name}
USING ({                  } ...)
       {'control-literal' }
```

Control-field-name or *'control-literal'* identifies the control data item. The length and code system of the control data item must match that in the database.

```
[SHARE|SHR    ]
[             ]
[EXCLUSIVE|EXC]
```

These optional parameters determine the type of lock to be placed on the object record.

IDMS FINISH Statement

The IDMS FINISH statement signs off the database management system.

Syntax

```
IDMS FINISH
```

IDMS GET Statement

The IDMS GET statement retrieves current data records.

Syntax

```
         [       {record-field-name}]
IDMS GET [RECORD {                 }]
         [       {'record-literal' }]
```

Parameters

```
[       {record-field-name}]
[RECORD {                  }]
[       {'record-literal'  }]
```

Record-field-name or *'record-literal'* identifies the record to retrieve. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value. The default record is the current record type of the run-unit.

IDMS IF Statement

The IDMS IF statement tests the status of a set.

Syntax

```
                            {MEMBER  }
             {set-field-name} {NOMEMBER}
 IDMS IF SET {              } {        }
             {'set-literal' } {EMPTY   }
                            {NOEMPTY }
```

Parameters

```
    {set-field-name}
SET {              }
    {'set-literal' }
```

Set-field-name or *'set-literal'* identifies the set to be tested. *Set-field-name* must be a 16-byte alphanumeric field. *'Set-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

```
{MEMBER  }
{NOMEMBER}
{        }
{EMPTY   }
{NOEMPTY }
```

This parameter determines the type of test:

- Specify MEMBER if the current record is a member of specified set
- Specify NOMEMBER if the current record is not a member of specified set
- Specify EMPTY if no member record occurrences exist
- Specify NOEMPTY if member record occurrences exist.

Usage Notes

The IDMS IF statement is similar to the CA-Easytrieve IF statement in that a corresponding END-IF statement is required and the ELSE statement is optional. See the IF, ELSE-IF, ELSE, and END-IF Statements later in this chapter for more information.

IDMS KEEP Statement

The IDMS KEEP statement places a shared or exclusive lock on a record.

Syntax

```
          [{RECORD} {field-name}]
IDMS KEEP [{SET   } {          }] [EXCLUSIVE]
          [{AREA  } {'literal' }]
```

Parameters

```
[{RECORD} {field-name}]
[{SET   } {          }]
[{AREA  } {'literal' }]
```

Field-name or *'literal'* identifies the desired record, set, or area to be locked. *Field-name* must be a 16-byte alphanumeric field. *'Literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

[EXCLUSIVE] This optional parameter controls the lock for the record. The default is SHARED LOCK.

IDMS MODIFY Statement

Format 1 of the IDMS MODIFY statement updates a record within the database. Format 2 updates a logical record within the database.

Syntax

Format 1

```
                   {record-field-name}
IDMS MODIFY RECORD {                 }
                   {'record-literal' }
```

Format 2

```
IDMS MODIFY RECORD logical-record-name [WHERE (boolean-expression)]
```

Parameters

Format 1

```
       {record-field-name}
RECORD {                  }
       {'record-literal' }
```

Record-field-name or *'record-literal'* identifies the record to be modified. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

Format 2

RECORD *logical-record-name* *Logical-record-name* is a one to sixteen-character name that identifies the logical record to be modified by CA-IDMS.

[WHERE (*boolean-expression*)] Code the optional WHERE clause to provide a Boolean expression to CA-IDMS to select the logical records to be modified.

IDMS OBTAIN Statement

The IDMS OBTAIN statement is used to retrieve logical records.

Note: To retrieve database records, see the IDMS FIND/OBTAIN Statement, described separately in this chapter.

Syntax

```
            {FIRST}
IDMS OBTAIN {     } RECORD logical-record-name [WHERE(boolean-expression)]
            {NEXT }
```

Parameters

```
{FIRST}
{     }
{NEXT }
```

Specify FIRST to retrieve the first occurrence of the logical record. Specify NEXT to retrieve subsequent occurrences.

RECORD *logical-record-name* *Logical-record-name* is a one to sixteen-character name that identifies the logical record to be retrieved. *Logical-record-name* must be the name of a logical record defined by a LOGICAL-RECORD statement.

`[WHERE (`*`boolean-expression`*`)]` Code the optional WHERE clause to provide a Boolean expression CA-IDMS uses to select the logical records to be retrieved.

IDMS READY Statement

The IDMS READY statement establishes area availability with the database manager.

Syntax

```
           [     {area-field-name}] [{RETRIEVAL} [PROTECTED]]
IDMS READY [AREA {                }] [{          } [          ]]
           [     {'area-literal' }] [{UPDATE    } [EXCLUSIVE]]
```

Parameters

```
[     {area-field-name}]
[AREA {                }]
[     {'area-literal' }]
```

Area-field-name or *'area-literal'* identifies the area to be made available for processing. *Area-field-name* must be a 16-byte alphanumeric field. *'Area-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value. If not specified, all areas in the subschema are readied.

```
[{RETRIEVAL} [PROTECTED]]
[{          } [          ]]
[{UPDATE    } [EXCLUSIVE]]
```

These optional parameters determine the type of access. The default is RETRIEVAL.

IDMS RETURN Statement

The IDMS RETURN statement retrieves the database key for an indexed record without retrieving the record itself.

Syntax

```
                                           {set-field-name}
IDMS RETURN DBKEY receive-field-name FROM {                }  +
                                           {'set-literal' }

           [KEY symbolic-key]    +

           {                          }
           {CURRENCY                  }
           {FIRST [CURRENCY]          }
           {LAST [CURRENCY]           }
```

```
{NEXT [CURRENCY]            }
{PRIOR [CURRENCY]           }
{                           }
{       {key-field-name}    }
{USING ({               }...)}
{       {'key-literal' }    }
```

Parameters

DBKEY *receive-field-name* This is a four-byte binary field with zero (0) decimal places. This field receives the DBKEY of the indexed record.

```
     {set-field-name}
FROM {              }
     {'set-literal' }
```

Set-field-name or *'set-literal'* identifies the index set to be accessed. *Set-field-name* must be a 16-byte alphanumeric field. *'Set-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

[KEY *symbolic-key*] This parameter retrieves the record's symbolic key into *symbolic-key*. *Symbolic-key* is the name of an alphanumeric field that is large enough to contain the record's symbolic key.

```
{                           }
{CURRENCY                   }
{FIRST [CURRENCY]           }
{LAST [CURRENCY]            }
{NEXT [CURRENCY]            }
{PRIOR [CURRENCY]           }
{                           }
{       {key-field-name}    }
{USING ({               }...)}
{       {'key-literal' }    }
```

These parameters determine the record for which the database key is returned:

- CURRENCY— the current index entry
- FIRST [CURRENCY]—the first entry in the index
- LAST [CURRENCY]— the last entry in the index
- NEXT [CURRENCY]— the entry following current of index. If the current of index is the last entry, an error status of 1707 (END OF INDEX) is returned.
- PRIOR [CURRENCY]— the entry before current of index
- USING— the first index entry whose symbolic key matches the *key-field-name* or *'key-literal'*. If no such entry exists, a status code of 1726 (INDEX ENTRY NOT FOUND) is returned. The attributes of *key-field-name* or *'key-literal'* must match the symbolic key of the index.

IDMS ROLLBACK Statement

The IDMS ROLLBACK statement requests recovery.

Syntax

```
IDMS ROLLBACK [CONTINUE]
```

Parameters

[CONTINUE] This optional parameter specifies the action taken after the recovery. The default is to terminate the run-unit.

IDMS STORE Statement

Format 1 of the IDMS STORE statement places a new record occurrence into the database. Format 2 places a new logical record occurrence into the database.

Syntax

Format 1

```
                        {record-field-name}
     IDMS STORE RECORD {                  }
                        {'record-literal' }
```

Format 2

```
     IDMS STORE RECORD logical-record-name [WHERE (boolean-expression)]
```

Parameters

Format 1

```
       {record-field-name}
RECORD {                  }
       {'record-literal' }
```

Record-field-name or *'record-literal'* identifies the record to store. *Record-field-name* must be a 16-byte alphanumeric field. *'Record-literal'* must be alphanumeric and is padded to the right (if necessary) to create a 16-byte value.

Format 2

RECORD *logical-record-name* *Logical-record-name* is a one to sixteen-character name that identifies the logical record that CA-IDMS stores.

[WHERE (*boolean-expression*)] Code the optional WHERE clause to provide a Boolean expression CA-IDMS uses to select the logical records to be stored.

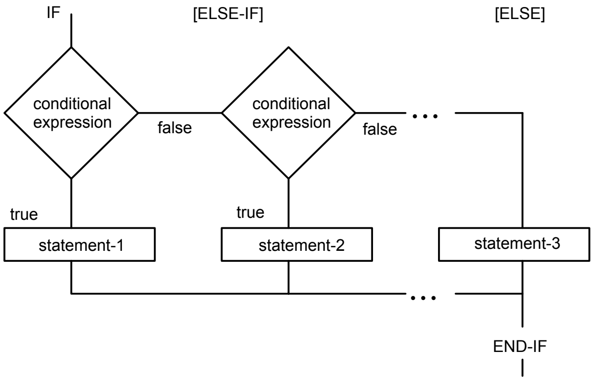
IF, ELSE-IF, ELSE, and END-IF Statements

The IF statement controls the execution of its associated statements. Associated statements are those that are coded between IF and END-IF.

Syntax

```
IF  conditional-expression-1
        [statement-1]

[ELSE-IF  conditional-expression-2]  [ . . . ]
[             [statement-2]        ]  [       ]

[ELSE            ]
[  [statement-3] ]

END-IF
```

The following diagram illustrates IF, ELSE-IF, ELSE, and END-IF logic:



Parameters

conditional-expression See Conditional Expressions for conditional expression syntax.

ELSE-IF ELSE-IF is optional and identifies a conditional expression to be tested when the previous conditional expression is false. ELSE-IF statements allow multiple conditions to be nested without requiring an END-IF statement or each condition. You can code as many ELSE-IF statements as necessary.

`ELSE` ELSE is optional and identifies the statements to be executed when conditions are false. When the conditions of the preceding IF or ELSE-IF are not satisfied, CA-Easytrieve continues execution with the statement following ELSE.

Note: ELSE must be on a source statement by itself unless it is followed by a period and a space.

`END-IF` END-IF terminates the logic associated with the previous IF statement. An END-IF statement must be specified after each IF statement and its associated statements. You do not specify an END-IF for an ELSE-IF.

Usage Notes

The truth value of the *conditional-expression-1* determines whether statement-1 is executed. CA-Easytrieve executes statements designated by statement-1 when *conditional-expression-1* is true. When *conditional-expression-1* is false, CA-Easytrieve tests *conditional-expression-2* if ELSE-IF is specified.

If ELSE-IF is specified, the truth value of *conditional-expression-2* determines whether statement-2 is executed. CA-Easytrieve executes statements designated by statement-2 when *conditional-expression-2* is true. When *conditional-expression-2* is false, CA-Easytrieve tests the conditional expression of the next ELSE-IF, if specified. If the last ELSE-IF statements conditional expression is also false, CA-Easytrieve executes statements designated by statement-3. You can nest as many ELSE-IF statements within the IF as necessary. You must terminate the IF statement with a single END-IF.

If ELSE-IF is not specified and *conditional-expression-1* is false, CA-Easytrieve executes statements designated by statement-3.

If the ELSE statement is not specified and the conditional-expression is false, no statements are executed and control passes to the statement following END-IF.

Statement-1, statement-2, and statement-3 each represent any number of CA-Easytrieve statements. Whenever one or more of these statements is an IF statement, the IF statements are considered to be nested. The format of nested IF statements is that statement-1, statement-2, and statement-3 of any IF can be an IF statement.

Examples

The following three examples illustrate the IF statement usage. In each of the illustrated cases, the field XMAS-BONUS is computed to be three or five percent over PAY-GROSS. When the field PAY-GROSS is non-numeric, a warning message is issued and the record is bypassed from further processing.

Example 1, without nested IF statements:

```
FILE PERSNL FB(150 1800)
%PERSNL
XMAS-BONUS       W 4 P 2 VALUE 0
TOT-XMAS-BONUS   W 6 P 2 VALUE 0
*
JOB INPUT PERSNL NAME MYPROG FINISH FINISH-PROC
   IF PAY-GROSS NOT NUMERIC
     DISPLAY EMP# '  PERSONNEL RECORD IS DAMAGED'
     GO TO JOB
   END-IF
   IF PAY-GROSS > 500.00
     XMAS-BONUS = PAY-GROSS * 1.03
   ELSE
     XMAS-BONUS = PAY-GROSS * 1.05
   END-IF
   TOT-XMAS-BONUS = TOT-XMAS-BONUS +
                    + XMAS-BONUS
   PRINT MYREPORT
*
FINISH-PROC. PROC
DISPLAY
DISPLAY 'TOTAL $ SPENT IN BONUS ' +
        'MONEY ====> ' TOT-XMAS-BONUS
END-PROC
*
REPORT MYREPORT
LINE NAME-LAST XMAS-BONUS
```

Example 2, with nested IF statements:

```
FILE PERSNL FB(150 1800)
%PERSNL
XMAS-BONUS      W 4 P 2 VALUE 0
TOT-XMAS-BONUS  W 6 P 2 VALUE 0
*
JOB INPUT PERSNL NAME MYPROG FINISH FINISH-PROC
   IF PAY-GROSS NOT NUMERIC
     DISPLAY EMP# '  PERSONNEL RECORD IS DAMAGED'
     GOTO JOB
   ELSE
      IF PAY-GROSS > 500.00
        XMAS-BONUS = PAY-GROSS * 1.03
      ELSE
        XMAS-BONUS = PAY-GROSS * 1.05
      END-IF
   END-IF
   TOT-XMAS-BONUS = TOT-XMAS-BONUS + XMAS-BONUS
   PRINT MYREPORT
*
FINISH-PROC. PROC
DISPLAY
DISPLAY 'TOTAL $ SPENT IN BONUS ' +
        'MONEY ====> ' TOT-XMAS-BONUS
END-PROC
```

```
*
REPORT MYREPORT
LINE NAME-LAST XMAS-BONUS
```

Example 3, with ELSE-IF statements:

```
FILE PERSNL FB(150 1800)
%PERSNL
XMAS-BONUS      W 4 P 2 VALUE 0
TOT-XMAS-BONUS  W 6 P 2 VALUE 0
*
JOB INPUT PERSNL NAME MYPROG FINISH FINISH-PROC
   IF PAY-GROSS NOT NUMERIC
     DISPLAY EMP# '  PERSONNEL RECORD IS DAMAGED'
     GOTO JOB
   ELSE-IF PAY-GROSS > 500.00
        XMAS-BONUS = PAY-GROSS * 1.03
   ELSE
        XMAS-BONUS = PAY-GROSS * 1.05
   END-IF
   TOT-XMAS-BONUS = TOT-XMAS-BONUS + XMAS-BONUS
   PRINT MYREPORT
*
FINISH-PROC. PROC
DISPLAY
DISPLAY 'TOTAL $ SPENT IN BONUS ' +
        'MONEY ====> ' TOT-XMAS-BONUS
END-PROC
*
REPORT MYREPORT
LINE NAME-LAST XMAS-BONUS
```

INITIATION Screen Procedure

An INITIATION procedure is invoked once during the start of the screen activity.

Syntax

```
INITIATION. PROC
```

Usage Notes

You use INITIATION to perform actions that are to be executed only once. Typically you use INITIATION to initialize a field or position a file at a specific starting position.

REFRESH and RESHOW are invalid in an INITIATION procedure.

If GOTO SCREEN is executed in an INITIATION procedure, the INITIATION procedure is terminated and the BEFORE-SCREEN procedure is invoked.

An INITIATION procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

```
INITIATION. PROC
  MESSAGE 'Enter an employee number.' LEVEL INFORMATION
  MOVE ZERO TO EMP-NO
  MOVE SPACES TO EMPNAME
END-PROC
```

INSERT Statement

The INSERT statement is used to insert a row into a CA-Easytrieve SQL file.

Syntax

```
INSERT [INTO] file-name
```

Parameters

[INTO] Optionally, code INTO for statement readability.

file-name *File-name* is a CA-Easytrieve SQL file.

Usage Notes

INSERT does not require an open cursor. If a cursor for the file is not open, one is not opened automatically. If a cursor is open, the inserted record does not appear in the cursor's result set until the cursor is closed and re-opened with a new SELECT statement.

The file must be specified with the UPDATE parameter.

Example

The following example inserts a new row into a table.

```
FILE PERSNL SQL (PERSONNEL) UPDATE
EMPNAME       *   20  A
WORKDEPT      *   2   P   0
EMPPHONE      *   3   P   0
PROGRAM NAME RETRIEVE-PERSONNEL
  EMPNAME = 'WIMN    GLORIA'
  WORKDEPT = 921
  EMPPHONE = 3478
  INSERT INTO PERSNL
```

JOB Statement

The JOB statement defines and initiates a processing activity. In a JOB activity, statements can specify various processing tasks:

- Retrieval of input files and databases
- Examination and manipulation of data
- Initiation of printed reports
- Production of output files and databases.

Syntax

```
JOB +

[      {(file-name [KEY field-name-1...)]...)} ]
[INPUT {NULL                                 } ] +
[      {SQL                                  } ]

[START start-proc-name] +

[FINISH finish-proc-name] +

[NAME job-name] +

[        [ACTIVITY  ] [TERMINAL  ] ]
[COMMIT ([          ] [          ])]
[        [NOACTIVITY] [NOTERMINAL] ]
```

Parameters

`[INPUT]` The optional INPUT parameter identifies the automatic input to the activity.

When you do not specify INPUT, CA-Easytrieve automatically provides an input file. If a SORT activity immediately preceded the current JOB activity, the default input is the output file from that SORT activity. Otherwise, the default input is the first file named in the library section.

`{file-name}` *File-name* identifies the automatic input file. *File-name* identifies any file defined in the library section of the program eligible for sequential input processing. When CA-Easytrieve processes the last automatic input record, CA-Easytrieve terminates the job activity.

Note: Except in CICS, CA-Easytrieve issues a GET HOLD for a VSAM file with the UPDATE parameter as automatic input. This allows you to update an automatic input file in all environments except CICS.

Coding an SQL *filename* in the JOB statement causes an SQL cursor to be opened at the start of the job activity. An SQL FETCH command or statement is executed for each execution of the JOB statement (including the first). See the "SQL Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

Coding a CA-IDMS file name on the JOB statement requires a RETRIEVE or SELECT statement to follow. See the "CA-IDMS Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

Coding an IMS/DLI file name on the JOB statement requires a RETRIEVE statement to follow. See the "IMS/DLI Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

{[KEY (*key-field-name* ...)]} Specify key fields for JOB statement activities.

Code *KEY (key-field-name)* for each *file-name* of a synchronized file input process. During synchronized file processing, CA-Easytrieve sequentially processes the file(s) using KEY fields. KEY fields can be any fields from the associated file. The only exceptions are varying length fields, which cannot be used as keys. For more detailed information about synchronized file processing, see the *CA-Easytrieve Programmer Guide*.

Note: Synchronized file processing is not allowed for CA-IDMS and IMS/DLI database files.

{NULL} Code NULL as a file-name to inhibit automatic input. Use this when no input is required or when input is retrieved by statements in the activity. When using NULL, a STOP or TRANSFER statement must be executed in the JOB activity, otherwise the activity executes indefinitely.

{SQL} Code SQL instead of a *filename* to use automatic retrieval of an SQL database without a file. The selection criteria for the input is specified on the non-file SQL SELECT statement that must immediately follow the JOB statement. See the "SQL Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information about automatic retrieval without a file.

[START *start-proc-name*] The optional START *start-proc-name* parameter identifies a procedure to be executed during the initiation of the JOB.

CA-Easytrieve optionally performs the procedure coded in *start-proc-name* before it retrieves the first automatic input record. A typical START procedure sets working storage fields to an initial value or positions a file to a specific record. You cannot reference fields in automatic input files because no records have been retrieved at this stage of processing.

If GOTO JOB is executed in a START procedure, the START procedure is terminated.

[FINISH *finish-proc-name*] The optional FINISH *finish-proc-name* parameter identifies a procedure to be executed during the normal termination of the JOB. After CA-Easytrieve processes the last automatic input record, it performs the *finish-proc-name* procedure. A typical *finish-proc-name* procedure displays control information accumulated during the activity.

If GOTO JOB is executed in a FINISH procedure, the FINISH procedure is terminated at that point.

[NAME *job-name*] The NAME parameter names the JOB activity. *Job-name* can:

- Be up to 128 characters in length
- Contain any character other than a delimiter
- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

This parameter is used for documentation purposes or to identify this JOB on an EXECUTE statement.

```
[        [ACTIVITY  ]  [TERMINAL  ] ]
[COMMIT ([          ]  [          ])]
[        [NOACTIVITY]  [NOTERMINAL] ]
```

Specify the COMMIT parameter to control the logical unit of work. COMMIT indicates when the activity commits recoverable work. Each commit point posts all updates, additions and deletions, terminates holds, and closes SQL cursors.

Specify ACTIVITY to commit all recoverable work during the normal termination of the activity. Specify NOACTIVITY to tell CA-Easytrieve not to commit at the end of the activity. NOACTIVITY is the default.

Specify TERMINAL to commit all recoverable work during any terminal I/O operation. In CICS, this results in terminal I/O being performed in a pseudo-conversational mode. Specify NOTERMINAL to tell CA-Easytrieve not to commit during a terminal I/O. TERMINAL is the default.

If this activity is executed by an activity that has NOTERMINAL specified, this activity performs terminal I/O as if NOTERMINAL was specified.

Note: You can also issue your own COMMIT and ROLLBACK statements to commit or recover work on a controlled basis.

See the *CA-Easytrieve Programmer Guide* for more information.

Usage Notes

The JOB statement can also identify the name of an automatic input file (which can be any file or database that is processed sequentially).

JOB activities can be EXECUTEd by PROGRAM or SCREEN activities. If a PROGRAM activity is not coded, JOB and SORT activities are automatically executed sequentially until a SCREEN activity is encountered.

The following example illustrates the position of the JOB activities in a CA-Easytrieve program.

```
Environment
...
Library
...
Activities                 {JOB...
...                        {...
JOB, SCREEN     <----------{  job procedures
and/or                     {  ...
SORT                       {  ...
...                        {  reports
...                        {  ...
```

See the *CA-Easytrieve Programmer Guide* for a discussion of JOB Activity Flow Control.

Examples

The first example illustrates a JOB statement that automatically reads a sequential file.

```
JOB INPUT PERSNL NAME SCAN-PERSONNEL-RECORDS
```

The second example illustrates synchronized file processing. It shows a JOB statement for matching a transaction file (TRANFILE) with a master file (MASTFILE). The JOB uses the FINISH parameter to execute a procedure when all the input records have been recorded.

```
JOB NAME MATCH-FILES INPUT (TRANFILE KEY TRAN-EMP-NO  +
                            MASTFILE KEY MAST-EMP-NO) +
    FINISH DISPLAY-EOJ-MESSAGE
```

KEY Statement

The KEY statement is used to:

- Define valid keys for a screen
- Specify descriptive text to be displayed for each valid key
- Assign automatic functions to be executed for each valid key.

Syntax

```
                                                   [EXIT   ]
KEY key-name [THRU key-name]...[NAME 'literal'] [       ] [IMMEDIATE]
                                                   [REFRESH]
```

Parameters

key-name Specify a symbolic name for a terminal key as described by the system-defined field, KEY-PRESSED.

KEY-PRESSED is a two-byte binary field that contains a value representing the most recent terminal key pressed by the terminal user.

CA-Easytrieve automatically defines symbolic names that correspond to values for the most common keys.

Terminal Key	Symbolic Name	Constant Value
Enter	ENTER	1
Clear	CLEAR	11
PA1 thru PA3	PA1 thru PA3	12 thru 14
PF1 thru PF24	F1 thru F24	21 thru 44
F1 thru F12	F1 thru F12	21 thru 32

Note: Only terminal keys with a KEY-PRESSED symbolic name can be used on a KEY statement. If other terminal keys (such as test request) are required, you must test KEY-PRESSED using the constant value of the terminal key in your program code. If you test for terminal keys without a symbolic name, you cannot code KEY statements in your program.

[THRU *key-name*] Use THRU *key-name* to specify a range of *key-names*. A range of *key-names* includes all keys whose constant values for KEY-PRESSED fall between the constant values of the keys you specify for the range. For example, if you code:

```
KEY CLEAR THRU F12
```

the PA1, PA2, and PA3 keys are also valid. The constant values of the PA keys (12, 13, 14) fall between the value for CLEAR (11) and F12 (32).

Note: You can also specify a series of non-consecutive *key-names* by omitting THRU. You can optionally separate a series of *key-names* with commas for readability. You can specify a range of *key-names* and a series of *key-names* on the same KEY statement. See the examples below.

[NAME '*literal*'] The optional NAME parameter allows you to specify descriptive text to be displayed with the key on the screen. The format displayed on the screen is:

key-name=literal

For example:

```
F1=Help  F3=Exit  F12=Cancel
```

'Literal' can contain a maximum of 20 characters.

To display only the *key-name* on a screen, code NAME *'literal'* with a blank space between single quotes (' ').

If you do not code NAME, no display is created for the key.

```
[EXIT   ]
[REFRESH]
```

Optionally, you can code EXIT or REFRESH to specify the branch action taken when a user presses *key-name*. If EXIT or REFRESH is specified, the action is automatically executed by CA-Easytrieve and the AFTER-SCREEN procedure (if any) is not executed.

Specify EXIT to terminate the screen activity after editing and extracting data from screen fields into program fields.

Specify REFRESH to restore the initial screen image by rebuilding it with current values of the program fields. Data in screen fields is edited and extracted into program fields.

If an action is not specified for *key-name*, you can test for *key-name* in your SCREEN activity procedures with the system-defined field, KEY-PRESSED.

[IMMEDIATE] Specify IMMEDIATE to execute a branch action, or the AFTER-SCREEN procedure if no action is specified, without editing data in screen fields and moving it into the program fields.

Usage Notes

If a key that is not defined on a KEY statement is pressed, an error message is displayed on the terminal prompting the user to press a valid key.

If no KEY statements are coded, all keys are valid and you must provide code for all keys in your SCREEN activity procedures.

The function key area is built depending on the sequence of keys specified in KEY statements. You must specify keys in the order you want them displayed.

The key display area is built on the bottom line of a screen. If the key display area requires additional lines because of the number of keys and the length of the descriptive text, additional lines at the bottom of the screen are used.

When used as the result of pressing an IMMEDIATE key, REFRESH re-displays the screen image with the original data displayed on the screen. This is useful when the terminal user enters erroneous data on the screen and wants to restore the screen with its original data.

When used as the result of a non-IMMEDIATE key, REFRESH can be used to rebuild the screen using current data from the screen.

REFRESH can also be invoked by using the REFRESH statement. See the REFRESH Statement for more information.

EXIT can also be invoked indirectly by executing it in a screen procedure. See the EXIT Statement for more information.

Note: If you specify that one or more message areas use the same screen row as the function key area, messages might overlap the function key area. The default for the message area is the row immediately preceding the key display area.

Note: The CLEAR, PA1, PA2, and PA3 keys do not transmit data from the screen to the program. Also, cursor positioning cannot be ascertained when these keys are pressed.

Examples

The following table shows KEY statement examples:

Code	Meaning
KEY F1	F1 is valid, but nothing is displayed on the screen. You must provide code.
KEY F1 THRU F24	F1 through F24 are valid keys, but nothing is displayed on the screen. You must provide code for all keys.
KEY F1 NAME ‘Help’	F1 is valid. F1=Help is displayed on the screen. You must provide code.
KEY F1 F4	F1 and F4 are valid keys, but nothing is displayed on the screen. You must provide code.
KEY F1 THRU F4, F8	F1, F2, F3, F4, and F8 are valid keys, but nothing is displayed on the screen. You must provide code for all keys.
KEY F12 EXIT NAME + ‘CANCEL’ IMMEDIATE	F12 terminates the screen activity without moving data from screen fields into program fields. The AFTER-SCREEN

Code	Meaning
	procedure (if any) is not executed. F12=CANCEL is displayed on the screen.
KEY F3 IMMEDIATE	The AFTER-SCREEN procedure (if any) is executed without editing or moving data in screen fields to program fields. Nothing is displayed on the screen. You must provide code for F3.
KEY F3 EXIT	F3 terminates the activity after editing and moving data from the screen.
KEY F5 IMMEDIATE + REFRESH NAME + 'Refresh'	F5 ignores the data on the screen and rebuilds the screen with the values currently in memory. F5=Refresh is displayed on the screen.

LINE Statement

The LINE statement defines the contents of a report line. One or more field values or literals can be contained on a report line; each one is a line item. The data format of the field or literal remains unchanged.

Syntax

```
                   {[            ] field-name }
                   {[#font-number]            }
                   {[            ] 'literal'  }
LINE [line-number] {+offset                   } ...
                   {-offset                   }
                   {COL column-number         }
                   {POS position-number       }
```

Parameters

[*line-number*] Specify the optional line number with *line-number*. The line number specifies the position of the line in the line group. The value must be from 1 to 99; the default is 1. You can omit *line-number* for the first LINE. You must specify the line numbers for multiple LINE statements in ascending order with no duplicates. Specify at least one data item (*field-name* or *literal*) on each LINE statement.

[#*font-number*] (Mainframe and UNIX only) #*Font-number* identifies the font specifications to be used for the next display item. You can only specify this option if the report has been associated with an extended reporting printer. #*Font-number* identifies the number of a font defined for the associated extended reporting printer. If you do not code the font, the next display item uses the default font for the assigned extended reporting printer.

{*field-name*} *Field-name* can specify any field contained in an active file or in working storage. If the field is contained in a file or W storage, data is transferred to the print line at the time the PRINT statement is executed. If the field is contained in S storage, data is transferred to the print line at the time the line is printed.

Note: *Field-name* cannot specify a K field.

{'*literal*'} *'Literal'* defines a static value for a line item. It must be a numeric literal, hexadecimal literal, or an alphanumeric literal. Alphanumeric literals must be enclosed within single quotes.

```
{+offset}
{       }
{-offset}
```

The space adjustment parameters, +*offset* or -*offset*, modify the spacing between line items. The *offset* value is added to or subtracted from the SPACE value on the REPORT statement to give the absolute spacing between line items. The absolute space value can range from zero to any amount that still allows the next line item to fit in the line defined by LINESIZE on the REPORT statement.

{COL *column-number*} COL specifies the column number where the next line item is placed. The value of *column-number* has a valid range of 1 to 'nnn,' where 'nnn' cannot be so large that the following line item extends beyond the end of the line defined by LINESIZE.

Note: You must specify the NOADJUST parameter on the REPORT statement to use the COL parameter.

When the report is associated with an extended reporting printer, an error results if two or more fields and/or literals overlap.

{POS *position-number*} The POS parameter enables you to position line items on lines 2 through 99 so that they line up under particular line items on the first line. *Position-number* corresponds to the line item number of LINE 01 under which the line item is placed.

Usage Notes

For control reports, any quantitative field on the LINE statement is automatically totaled on each summary line. This feature can be overridden on the SUM statement.

Example

```
LINE 1  REGION BRANCH +5 DEPT EMPNAME
LINE 2  POS 4 ADDRESS
LINE 3  'NET==>' -2 PAY-NET POS 4 CITY ST ZIP
```

LINK Statement

The LINK statement is used to transfer control from the current program (parent program) to another named program (child program). When the child program terminates, execution is then returned to the statement following the LINK statement in the parent program.

Syntax

```
     {program-field-name}
LINK {                  } +
     {'program-name'    }

     [      {field-name}]
     [USING {          }] +
     [      {'literal' }]

     [GIVING field-name] +

     [HOST] +

     [WAIT wait-time]  +

     [NOENTER]
```

Parameters

```
{program-field-name}
{                  }
{'program-name'    }
```

Program-field-name is the name of the field that contains the name of the program to which you want to LINK.

'Program-name' is the name of the program to which you want to LINK.

```
      {field-name}
USING {          }
      {'literal' }
```

Code USING to pass a single parameter to the child program.

Field-name is the name of a field containing the parameter you want to pass to the child program.

'Literal' is a literal value you want to pass to the child program.

`[GIVING field-name]` Specify GIVING to indicate that the parent program can accept a return parameter from the child program. *Field-name* is the name of a field to which the returned parameter is written. See the *CA-Easytrieve Programmer Guide* for more information.

Note: If the child program returns a value, but the GIVING parameter is not specified, the value is ignored. Not all operating systems allow the child program to return data to the parent program.

`[HOST]` (Workstation only) Specify HOST if you want to send a command to the mainframe. HOST sends the program name and the using field as an EBCDIC string directly to the host using an HLLAPI. You must have the appropriate hardware, emulator and HLLAPI to use this parameter. See the "Coding a CA-Easytrieve Program" chapter in the *CA-Easytrieve Programmer Guide* for more information.

`[WAIT wait-time]` (Workstation only) The WAIT parameter allows you to specify the time in seconds to wait before sending the command to the host. This parameter can be coded only if HOST has been specified.

Wait-time must be an integer from 0 to 32767.

`[NOENTER]` (Workstation only) The NOENTER parameter inhibits the automatic Enter (@E) sequence at the end of the command. This is useful if you want to send your own AID key sequence to the host. See the EEHLLAPI manual for more information about AID keys.

Usage Notes

LINK can be used to invoke any program written in any language that is supported by the operating system in which the program is executing, including CA-Easytrieve. Similarly, the program can issue any command supported by the operating system.

A program invoked using the LINK statement can issue terminal I/O or display reports, but only in fully-conversational mode. See the *CA-Easytrieve Programmer Guide* for more information.

Note: If you code the USING or GIVING parameter on the LINK statement, you must code a PROGRAM statement to handle the parameters in the child program when it is written in CA-Easytrieve.

Example

```
LINK 'PROGB' USING EMP# GIVING PROGB-RETURN
```

LIST Statement

LIST regulates the printing or suppression of all statements in the printed output.

Syntax

```
     [ON ] [MACROS  ]
LIST [   ] [        ]
     [OFF] [NOMACROS]
```

Parameters

```
[ON ]
[OFF]
```

ON specifies that all subsequent statements are to be printed. OFF suppresses the printing of all subsequent statements.

```
[MACROS  ]
[NOMACROS]
```

MACROS specifies that macro statements are to be printed if a LIST ON is in effect. NOMACROS suppresses the printing of macro statements.

The default is LIST ON MACROS.

Usage Notes

You can place a LIST statement anywhere in CA-Easytrieve source code. LIST must be on a record by itself.

LIST does not appear in the printed output.

To suppress all CA-Easytrieve listing information, use the following:

```
LIST OFF
PARM LIST(NOPARM)
```

See the PARM Statement for more information.

LOGICAL-RECORD Statement (CA-IDMS)

(Workstation only) Code LOGICAL-RECORD statements following the FILE statement to identify the logical records available for automatic or controlled processing of CA-IDMS databases.

Syntax

```
LOGICAL-RECORD   record-name
```

Parameters

record-name *Record-name* is the one to sixteen-character name of the logical record as defined in the subschema.

Usage Notes

The LOGICAL-RECORD statement cannot be used to define database records. To define a database record, use the RECORD statement. In addition, fields cannot be defined in association with the LOGICAL-RECORD statement. Fields are defined following the ELEMENT-RECORD statement.

You can use IDD statements to automatically generate LOGICAL-RECORD, ELEMENT-RECORD, and DEFINE statements.

MACRO Statement

The MACRO prototype statement must be the first statement of a macro. It optionally defines the parameters of a macro. Positional and keyword parameters can be used.

Syntax

```
MACRO [positional-count] +

      [positional-parameters] ... [keyword-parameters] ...
```

Parameters

MACRO MACRO must be the first word on a prototype statement.

`[positional-count]` *Positional-count* is an optional parameter that specifies the number of *positional-parameters* on the prototype statement. It is required only when you use *keyword-parameters* and *positional-parameters*. You must code the value as zero when you specify only *keyword-parameters* on the prototype statement.

`[positional-parameters]` Use *positional-parameters* when a value is always required for the parameters each time the macro is invoked. Frequently-used parameters are often positional, because you need only code the value of the parameter.

You must code *positional-parameters* before any *keyword-parameters*. The positional values are substituted according to their position on the prototype statement.

`[keyword-parameters]` Use *keyword-parameters*:

- To help keep track of a large number of parameters
- For optionally-used parameters
- To specify a default value for parameters.

Keyword-parameters have two parts: the keyword name and the default value.

Examples

The following series of examples depict the coding of macro prototype statements. See the "System Services" chapter in the *CA-Easytrieve Programmer Guide* for more information.

Macro with No Substitution Parameters

```
MACRO
...
...
```

Macro with only positional parameters

```
MACRO  POS1  POS2
...
...
```

The number of *positional-parameters* is not indicated. You could have coded the optional *positional-count* parameter as a '2.'

Macro with only keyword parameters

```
MACRO  0  KEY1 VALUE1  KEY2 VALUE2
...
...
```

Code the number of *positional-parameters* as zero. *Positional-count* is a required parameter when you use *keyword-parameters*.

Macro with positional and keyword parameters

```
MACRO 1  POS1  KEY1  VALUE1
...
...
```

Macros with both *positional* and *keyword-parameters* require that you supply *positional-parameters* first, and also supply a *positional-count*.

MASK Parameter

The optional MASK parameter establishes a pattern (edit mask) for a field name. The MASK parameter can be coded in the syntax of the following CA-Easytrieve statements:

- DEFINE
- ROW

Syntax

```
[MASK ({[mask-identifier][BWZ]['mask-literal']|HEX })]
```

[*mask-identifier*] Any letter from A through Y can be used as an optional *mask-identifier*. You can use the letter to identify a new mask or to retrieve a mask that was previously defined in the Site Options Table or by a mask parameter on a previous field definition. If the new mask that you identify does not already exist, CA-Easytrieve retains the mask for future reference. If you subsequently reference a *field-name* for display, CA-Easytrieve automatically uses the associated letter identifier to determine the edit mask. Do not use the same identifier to establish more than one mask. You can define 192 unidentified edit masks and 25 identified edit masks for a total of 217 edit masks.

[BWZ] The BWZ (blank when zero) option suppresses the display of *field-name* when it contains all zeros. BWZ can be used by itself or with other options on the MASK parameter.

['*mask-literal*'] *'Mask-literal'* defines an edit mask and must be enclosed within single quotes. The actual edit mask is coded according to the rules specified below under Editing Rules.

HEX HEX is a special edit mask that instructs CA-Easytrieve to display the contents of *field-name* in double-digit hexadecimal format. You can display fields of up to 50 bytes with the HEX mask.

Note: HEX edit masks are not allowed for VARYING fields.

Editing Rules

- CA-Easytrieve edits field data only at the time of display and according to a specified edit mask pattern.
- The MASK parameter of the DEFINE and ROW statements specifies the edit mask pattern.
- Each digit of the field must be designated in the mask by an edit mask character:

Symbol	Meaning
9	Prints a digit.
Z	Prints a digit, except for leading zeros.
*	Prints asterisks instead of leading zeros.
–	Prints a minus sign prior to the first non-zero digit of a negative number.
$	Prints a currency symbol prior to the first non-zero digit. The type of currency symbol ($, ¥, £, _, etc.) is determined by the MONEY Site Option.
x	Insertion symbol - prints any character with the edited data.

Decimal Digits

- When you display data, there is no implied relationship between the number of decimal digits in the edit mask and the number of decimal digits in the field definition. You must code the correct number of decimal digits in the mask.
- When screen data is edited against a mask, the decimal point is automatically aligned.

Alphanumeric Fields

- Alphanumeric fields cannot be edited. (The exception is MASK HEX.)

Currency Symbols

- The currency symbol indicator is recognized in the input edit mask and appears in the output edit mask. For example, if the currency symbol is set to #, a valid edit mask is '###,##9.99.'

Insertion Symbols

- Z, $, -, and * print digits only when coded as the first symbol of the edit mask, and only up to the first 9 symbols.

  All other symbols before the last digit position are treated as insertion symbols, including Z, $, -, and *. The symbols , (comma) and . (period) can also be used as insertion symbols.

  Insertion symbols before the first digit position always print.

- Insertion symbols between digit positions print according to the following rules:
  - If the symbol that prints a digit following the insertion symbol is a 9, the insertion characters always print.
  - If the digit position following the insertion symbols is a Z, $, -, or *, the insertion symbols print only if the digit position prints. If the digit position does not print, the insertion symbols are replaced by fill symbols.

For example, in the mask 'ZZZ,999.99,' the comma always prints. In the mask 'ZZZ,Z99,99,' the comma prints only if the digit prior to the comma is non-zero.

Fill Characters

- The default fill character for an edit mask is a blank, unless an * (asterisk) is specified.

Mask Display Length

- When the first symbol of an edit mask is a - (dash) or a currency symbol, the display length of the mask is the length of the mask plus one.
- The mask for a SUM field in a report is automatically increased by the number of digits specified by the SUMSPACE parameter on the REPORT statement. CA-Easytrieve duplicates the first digit position the required number of times.

Negative Indicators

- Symbols following the last digit position specify the negative indicator. The symbols print if the value edited is negative. If the value edited is positive, the symbols are replaced by fill characters.

System Default Masks - Numeric Fields

When you do not specify a mask, the following defaults apply:

```
 Number of
 Decimals                       Mask

   none            ZZZZZZZZZZZZZZZZZZ  *
     0        ZZZ,ZZZ,ZZZ,ZZZ,ZZZ,ZZZ-
     1         ZZ,ZZZ,ZZZ,ZZZ,ZZZ,ZZZ.9-
     2          Z,ZZZ,ZZZ,ZZZ,ZZZ,ZZZ.99-
     3            ZZZ,ZZZ,ZZZ,ZZZ,ZZZ.999-
     4             ZZ,ZZZ,ZZZ,ZZZ,ZZZ.9999-
     5              Z,ZZZ,ZZZ,ZZZ,ZZZ.99999-
     6                ZZZ,ZZZ,ZZZ,ZZZ.999999-
     7                 ZZ,ZZZ,ZZZ,ZZZ.9999999-
     8                  Z,ZZZ,ZZZ,ZZZ.99999999-
     9                    ZZZ,ZZZ,ZZZ.999999999-
    10                     ZZ,ZZZ,ZZZ.9999999999-
    11                      Z,ZZZ,ZZZ.99999999999-
    12                        ZZZ,ZZZ.999999999999-
    13                         ZZ,ZZZ.9999999999999-
    14                          Z,ZZZ.99999999999999-
    15                            ZZZ.999999999999999-
    16                             ZZ.9999999999999999-
    17                              Z.99999999999999999-
    18                               .999999999999999999-
* For zoned decimal fields with no decimals, the default mask
  is '999999999999999999'.
```

Your system administrator can define additional edit masks in the Site Options Table when CA-Easytrieve is installed.

Leading Zeros

CA-Easytrieve provides a number of methods for dealing with leading zeros by displaying, suppressing, or replacing them.

Displaying

When leading zeros are an important part of the number (such as social security numbers and part numbers) an edit mask that displays these zeroes is essential. Following are examples of edit masks that display leading zeros:

Mask	Field Contents	Displayed Results
999-99-9999	053707163	053-70-7163
(99)-9999	006421	(00)-6421

Suppressing

In some instances, leading zeros add unnecessary information and can confuse the reader. You can suppress the display of leading zeros by using one of the following masks:

Mask	Field Contents	Displayed Results
$$,$$9	01234	$1,234
$$,$$9	00008	$8
$$,$$9.99	0123456	$1,234.56
ZZZ,ZZ9	000123	123
---,--9	+001234	1,234
---,--9	-001234	-1,234

Replacing

In cases where fields need to be protected (such as check amounts), you can use edit masks that replace leading zeros with other symbols:

Mask	Field Contents	Displayed Results
**9	001	**1
,9	01234	*1,234
,9.99	0123456	*1,234.56

Negative Numbers

CA-Easytrieve displays the symbols used as negative number indicators to the right of the last digit of the negative data that you edit. You can use any symbols as negative number indicators, although the most typical indicators are the minus sign (-) and the credit indicator (CR). If the number is positive, CA-Easytrieve inhibits the display of these symbol(s); however, when the field contents turns negative, the negative number indicators are edited into the displayed output by CA-Easytrieve:

Mask	Field Contents	Displayed Results
ZZZ-	-123	123-

Mask	Field Contents	Displayed Results
ZZZ-	+123	123
ZZZ CR	-123	123 CR
ZZZ CR	+123	123
ZZZ IS MINUS	-123	123 IS MINUS

Examples

The following edit mask examples illustrate editing mask rules:

Mask	Field Contents	Displayed Results
`‘Z,ZZZ,ZZZ.99’`	`.01`	`.01`
`‘ZZHELLOZZ9.99’`	`123.01`	`123.01`
`‘ZZHELLOZZ9.99’`	`1234.01`	`1HELLO234.01`
`‘**HELLO**9.99’`	`11.01`	`********11.01`
`‘**HELLO**9.99’`	`123.99`	`*******123.99`
`‘**HELLO**9.99’`	`1234.99`	`*1HELLO234.99`
`‘$$99$$99.99’`	`1234.99`	`$02$$34.99`
`‘999Z999.99’`	`12345.99`	`012Z345.99`
`‘SSN 999-99-9999’`	`123456789`	`SSN 123-45-6789`
`‘ZZZ.99 MINUS’`	`12.45`	`12.45`

`‘ZZZ.99 MINUS’`	`-12.45`	`12.45`
		`MINUS`

`‘***.99 MINUS’`	`12.45`	`*12.45******`
`‘***.99 MINUS’`	`-12.45`	`*12.45` `MINUS`
`‘---.99’`	`123.45`	`123.45`
`‘---.99’`	`-123.45`	`-123.45`

MEND Statement

The MEND statement is an optional macro termination command used at the end of a macro. MEND is required at the end of an instream macro. See the MSTART Statement.

Syntax

```
MEND
```

Usage Notes

MEND must be coded on a line by itself.

MESSAGE Statement

The MESSAGE statement allows you to issue your own specific messages for a screen activity. You define the message type and specify the message text using the MESSAGE statement.

Syntax

```
MESSAGE {'literal' }
        {          } ... +
        {field-name}

    [      {INFORMATION} ]
    [LEVEL {WARNING    } ]
    [      {ACTION     } ]
```

Parameters

```
{'literal' }
{          }
{field-name}
```

Use *'literal'* to define the text you want displayed in the message. Use *field-name* to specify a field whose contents you want displayed as part of the message. A message can consist of a combination of *literals* and *field-names*.

The maximum length of a message is 130 characters. If the message exceeds the message area for the screen on which it is displayed, the message is truncated.

```
[      {INFORMATION} ]
[LEVEL {WARNING    } ]
[      {ACTION     } ]
```

Use LEVEL to specify the type of message you are defining.

INFORMATION messages typically inform a user that processing is proceeding normally.

WARNING messages tell the user that a potentially undesirable condition could occur or has occurred even though he can ignore the error.

ACTION messages are the most severe. They tell a user that an error has occurred and an action is required to correct the error before he can continue. ACTION is the default message level if no level is specified.

Usage Notes

You can code the MESSAGE statement in a screen procedure or in another activity.

You can determine where messages of a particular level are displayed on the screen by overriding the default message area on a DEFAULT statement. (The default message area is one line above the function key display area at the bottom of the screen.) You can also use the DEFAULT statement to override default message attributes.

CA-Easytrieve maintains an internal message area for each type of message. The MESSAGE statement updates the pending message area. When the next screen is displayed, the screen message area is built from the pending message.

If different levels of messages are displayed on the same line (by default or override), then the message displayed is controlled by message precedence. If two messages are sent to the same line on the screen, the message with the highest severity is displayed. The severity precedence from highest to lowest is:

- ACTION
- WARNING
- INFORMATION

If multiple MESSAGE statements of the same precedence are issued before displaying the screen, the last message issued is displayed. There are Site Options that determine the display attributes for the three levels of messages. You can override these attributes on a DEFAULT statement.

Example

```
MESSAGE 'Department of ' EMP-DEPT ' not 900-999.' LEVEL ACTION
```

MOVE Statement

MOVE transfers character strings from one storage location to another. The MOVE statement is especially useful for moving data without conversion and for moving variable length data strings.

Syntax

Format 1

```
       {send-file-name   }
       {send-record-name} [send-length-field  ]
  MOVE {                } [                   ]  +
       {send-field-name } [send-length-literal]
       {send-literal    }

      {receive-file-name  } [receive-length-field  ]
   TO {receive-record-name} [                      ] [FILL fill-character]
      {receive-field-name } [receive-length-literal]
```

Format 2

```
       {NULL  }
       {SPACE }
       {SPACES}
  MOVE {      } TO receive-field-name ...
       {ZERO  }
       {ZEROS }
       {ZEROES}
```

Parameters

Format 1

```
{send-file-name   }
{send-record-name}
{                }
{send-field-name }
{send-literal    }
```

The first parameter after the MOVE keyword (*send-file-name*, *send-record-name*, *send-field-name* or *send-literal*) identifies the sending data area. *Send-file-name* or *send-record-name* can be any file or database record with current data availability. When *send-file-name* is a CA-IDMS file, all records in the file are moved.

The default length of *send-file-name* is the current value of the system-defined RECORD-LENGTH field.

Note: If *send-literal* is non-numeric, it must be enclosed within single quotes.

```
[send-length-field  ]
[send-length-literal]
```

You can override the length of the sending field with the current value of *send-length-field* or *send-length-literal*.

```
{receive-file-name  }
```

```
{receive-record-name}
{receive-field-name }
```

The above parameters identify the receiving data area. *Receive-file-name* or *receive-record-name* can be any file or database record with current data availability. The default length of *receive-file-name* is the current value of the system-defined RECORD-LENGTH field.

```
[receive-length-field  ]
[receive-length-literal]
```

You can override the length of the receiving field with the current value of *receive-length-field* or *receive-length-literal*.

[FILL *fill-character*] CA-Easytrieve truncates longer sending fields on the right. Longer receiving fields are padded on the right with spaces or a character you specify in *fill-character*.

Fill-character must be one or two bytes. Non-numeric characters must be enclosed within single quotes. When *fill-character* contains numeric characters, they are treated as a zoned decimal value.

Format 2

```
{NULL  }
{SPACE }
{SPACES}
{      }
{ZERO  }
{ZEROS }
{ZEROES}
```

The first parameter after the MOVE keyword (NULL, SPACE, SPACES, ZERO, ZEROS, or ZEROES) identifies the sending data area. The default length of the field is the defined length of *receive-field-name*. Moving spaces or zeros to a field fills the entire field with the selected character. Moving nulls sets a nullable field to NULL. Moving spaces or zeros sets a nullable field to NOT NULL.

receive-field-name *Receive-field-name* identifies the receiving data area. Multiple *receive-field-names* can be specified. *Receive-field-names* are set to the appropriate data format, such as packed zero for fields with a type of P.

Usage Notes

When you specify Format 1 parameters, data moves from left to right as if both areas were alphanumeric. The data moved is unconverted. *Send-file-name* and *receive-file-name* can be any file in which data is currently available. See Assignment and Moves in the "Coding a CA-Easytrieve Program" chapter of the *CA-Easytrieve Programmer Guide* for MOVE statement specification rules.

When you process an SQL table as a CA-Easytrieve file, CA-Easytrieve knows which fields are nullable. This information is obtained automatically from the SQL catalog when used to generate CA-Easytrieve field definitions.

Examples

Move statement example 1

```
FILE PERSNL SQL (PERSONNEL)
SQL INCLUDE (EMP#) FROM PERSONNEL LOCATION *  NULLABLE
DEFINE CTR1  W   10 N
DEFINE CTR2  W    2 N
DEFINE PLINE W  130 A
. . .
MOVE ZEROS TO CTR1, CTR2
MOVE SPACES TO PLINE
MOVE NULL TO EMP#
```

Move statement example 2

```
Statements:

 DEFINE ASTERISK-LINE W 10 A  VALUE '=========='
 DEFINE COUNTER-1     W 10 N  VALUE 99
 DEFINE COUNTER-2     W  2 N  VALUE 66
 PROGRAM NAME MYPROG
    DISPLAY COUNTER-1 +2 COUNTER-2
    MOVE ZEROS TO COUNTER-1 COUNTER-2
    DISPLAY COUNTER-1 +2 COUNTER-2
    DISPLAY ASTERISK-LINE
    MOVE '*' TO ASTERISK-LINE FILL '*'
    DISPLAY ASTERISK-LINE

Results:

 0000000099   66
 0000000000   00
 ==========
 **********
```

MOVE LIKE Statement

The MOVE LIKE statement moves the contents of fields with identical names from one file, record, or working storage to another. Data movement and conversion follow the rules of the Assignment statement.

Syntax

```
          {send-file-name  }    {receive-file-name  }
MOVE LIKE {                } TO {                   }
          {send-record-name}    {receive-record-name}
```

Parameters

```
{send-file-name  }
{                }
{send-record-name}
```

Send-file-name or *send-record-name* identifies the sending data area.

```
{receive-file-name  }
{                   }
{receive-record-name}
```

Receive-file-name or *receive-record-name* identifies the receiving data area.

Usage Notes

When you issue a MOVE LIKE statement, the contents of fields in *send-file-name* or *send-record-name* replace the contents of fields with identical names in *receive-file-name* or *receive-record-name*. When *receive-file-name* or *receive-record-name* contains overlapping fields, the order in which the fields are defined is important. The moves occur starting with the first identically-named field in *receive-file-name* or *receive-record-name* and ending with the last identically-named field in the file.

Note: The order in which fields are processed differs from previous versions of CA-Easytrieve. In previous versions, the moves occurred starting with the last identically-named field in *receive-file-name* or *receive-record-name* and ended with the first identically-named field in the file.

If you want to move identically-named fields to or from working storage, you can use the keyword WORK as the *send-file-name* or *receive-file-name*.

CA-IDMS IDD Processing

In CA-IDMS IDD processing, the fields of a file defined by an IDD statement are organized into group item structures. A **group item** is a field subdivided by smaller fields. The smaller fields can themselves be group items and, therefore, subdivided by even smaller fields. A group item “owns” its subdividing fields. A field without subdivision is called an **elementary** field.

In IDD processing, MOVE LIKE assigns a new value to the receiving field if all of the following conditions are met:

- The sending and receiving fields have matching names
- The sending and receiving fields have matching qualifier (group item) names
- Either the sending or receiving field is an elementary field.

Record name qualifiers do not participate in the process of matching qualifiers between two fields. For example, in a MOVE LIKE from a record to a file (that owns the record), no matching is done between the record names of the receiving file and qualifiers of source (record) fields. Therefore, source fields can be matched to a field under one record and another source field can be matched to a field under a different record.

Differences Between MOVE LIKE and MOVE

The differences between the MOVE LIKE statement and the MOVE statement are as follows:

- The MOVE LIKE statement generates an Assignment statement that provides data conversion according to data type. See Assignment and Moves in the "Coding a CA-Easytrieve Program" chapter of the *CA-Easytrieve Programmer Guide* for Assignment statement specification rules.
- The MOVE statement moves data without converting it.

Example

```
FILE  PERSNL  FB(150   1800)
       REGION      1    1   N
       BRANCH      2    2   N
       NAME       17   16   A
            NAME-LAST      17  8  A
            NAME-FIRST     25  8  A
FILE  MYFILE FB(150  1800)
       COPY PERSNL
JOB  INPUT PERSONL NAME MYPROG
       MOVE LIKE PERSNL TO MYFILE
       PUT MYFILE
```

In the above example, MOVE LIKE generates the following Assignment statements:

```
MYFILE:NAME-FIRST = PERSNL:NAME-FIRST
MYFILE:NAME-LAST = PERSNL:NAME-LAST
MYFILE:NAME = PERSNL:NAME
MYFILE:BRANCH = PERSNL:BRANCH
MYFILE:REGION = PERSNL:REGION
```

Whatever values were in the fields of the file PERSNL are now found in the fields of the file MYFILE.

MSTART Statement

The MSTART statement is used to begin an instream macro. MSTART must be the first statement in the program.

Syntax

```
MSTART macro-name
```

Parameters

`macro-name` Specify the name of the macro. *Macro-name* must be from one to eight characters in length. The first character must be alphabetic.

Chapter

6 Statements N - R

NEWPAGE Statement

NEWPAGE is a listing control statement that ejects the printer to the top of the next page before printing the next line of the source program on the statement listing.

Syntax

```
NEWPAGE
```

Usage Notes

You can code a NEWPAGE statement anywhere in CA-Easytrieve source code. NEWPAGE must be on a record by itself. NEWPAGE does not appear in the printed output.

PARM Statement

The PARM statement allows you to override selected general standards for a program that are set in the Site Options Table. Alteration of the environment with the PARM statement lasts for only as long as the program is running.

Syntax

```
PARM   +

[       {SNAP   } ]
[ABEXIT {NOSNAP} ] +
[       {NO     } ]

[     {DYNAMIC    } ]
[BIND {STATIC-ONLY} ] +
[     {ANY        } ]

[     {STATIC ] ]
```

```
[CALL {       ] ] +
[     {DYNAMIC] ]

[             {EBCDIC        } ]
[CODE PROCESS {ASCII         } ] +
[             {dbcs-code-name} ]

[COMPILE] +

[        [CLIST  ] [PMAP  ] [DMAP  ] [FLDCHK  ] [FLOW  ]
[DEBUG ( [       ] [      ] [      ] [        ] [      ] +
[        [NOCLIST] [NOPMAP] [NODMAP] [NOFLDCHK] [NOFLOW]

                                          [STATE  ] [XREF   {LONG } ]   ]
        [FLOWSIZ number-of-table-entries] [       ] [       {     } ] ) ] +
                                          [NOSTATE] [NOXREF {SHORT} ]   ]

[LINK (program-name [R])] +

[     {PARM  } ]
[LIST {      } ] +
[     {NOPARM} ]

[PLAN (planname [command-program-name])] +

[PLANOPTS 'plan-options-module'] +

[PREPNAME (SQL-access-module ['access-userid'])] +

[SORT +

         [        {NO                    } ]
        ([ALTSEQ  {                      } ] +
         [        {(YES [alt-sort-table])} ]

        [DEVICE device-type] +

        [       {storage-amount           } ]
        [MEMORY {                         } ] +
        [       {(MAX [-storage-released])} ]

        [      {ALL      [CONSOLE] }   ]
        [      {         [PRINTER] }   ]
        [      {                   }   ]
        [MSG ( {CRITICAL [CONSOLE] } ) ] +
        [      {         [PRINTER] }   ]
        [      {DEFAULT            }   ]
        [      {NO                 }   ]

        [RELEASE core-storage-amount] +

        [WORK number-of-work-data-sets])] +

[SQLID 'auth-id'] +

[          {FULL   } ]
[SQLSYNTAX {PARTIAL} ] +
[          {NONE   } ]

[SSID 'ssid'] +

[SYNTAX] +

[TRANSID 'transid'] +

[USERID ('connect-userid' ['password'])] +

[                           [       {DISK  } ]   ]
[VFM ([buffer-core-storage] [DEVICE {      } ] ) ]  +
[                           [       {MEMORY} ]   ]

[           {YES}              ]
[WORKFILE ( {   } [BLOCKMAX] )]
[           {NO }              ]
```

Parameters

```
[       {SNAP   } ]
[ABEXIT {NOSNAP} ]
[       {NO     } ]
```

ABEXIT indicates the level of control exercised over program interrupt codes 1 through 11. SNAP prints a formatted dump of CA-Easytrieve storage areas along with an error analysis report. NOSNAP prints only an error analysis report. NO inhibits CA-Easytrieve interception of program interrupts. ABEXIT is ignored on the workstation.

```
[     {DYNAMIC    } ]
[BIND {STATIC-ONLY} ]
[     {ANY        } ]
```

BIND is an SQL-related parameter that identifies the type of SQL bind that you want for the execution of your application program. BIND is currently only used by the mainframe DB2 SQL interface. It is ignored in other environments.

BIND DYNAMIC results in the dynamic execution of the SQL statements in your program. Dynamic processing requires SQL statements to be dynamically "prepared" before they can be executed. The SQL interface controls the SQL environment and does not prepare SQL statements repeatedly unless a syncpoint has been taken.

BIND STATIC-ONLY indicates that your application program is to execute statically. This option requires the creation of a "static-command-program" that is then processed by the DB2 preprocessor. The DB2 preprocessor generates a DBRM and finally a PLAN. During the execution of your application program, the SQL interface processes the SQL statements in the "static-command-program." If any errors are found in the "static-command-program" or its PLAN, SQL processing is terminated.

BIND ANY indicates that a "static-command-program" is to be generated and a PLAN created, as with an option of STATIC-ONLY. However, if the SQL interface encounters any errors with the "static-command-program" or its PLAN during the execution of your application program, it switches to dynamic processing.

BIND STATIC-ONLY or BIND ANY requires a value for the PLAN and LINK parameters. PLAN specifies the name of the "static-command-program" and its DB2 PLAN name. LINK identifies the load module name of your link-edited CA-Easytrieve program. Your CA-Easytrieve application program must run as a link-edited program for static SQL processing.

Note: Regardless of the option you specify for the BIND parameter, your program is dynamically processed when being processed by the interpreter, that is, whenever the CHECK or RUN commands are executed.

DYNAMIC is the default mode of execution if no value is specified for the BIND parameter in the program or in the Options Table. Otherwise, the BIND value in the Options Table becomes the default. The following table illustrates the use of the BIND parameter with values specified in the Options Table.

BIND Parameter	**Value Specified in the Options Table**			
Value Specified	**blank**	**A**	**S**	**D**
No BIND parameter specified	BIND defaults to DYNAMIC	BIND defaults to ANY	BIND defaults to STATIC-ONLY	BIND defaults to DYNAMIC
ANY	ANY is the BIND parameter	ANY is the BIND parameter	Invalid - an error occurs	Invalid - an error occurs
STATIC-ONLY	STATIC-ONLY is the BIND parameter	STATIC-ONLY is the BIND parameter	STATIC-ONLY is the BIND parameter	Invalid - an error occurs
DYNAMIC	DYNAMIC is the BIND parameter	Invalid - an error occurs	Invalid - an error occurs	DYNAMIC is the BIND parameter

```
[     {STATIC } ]
[CALL {       } ]
[     {DYNAMIC} ]
```

CALL enables you to specify how subprograms referenced in CALL statements are linked to your CA-Easytrieve program. STATIC indicates that you want the subprogram to be linked with your CA-Easytrieve program. DYNAMIC indicates that you want the subprogram to be dynamically loaded. The default is STATIC. CALL is invalid on the Workstation and mainframe.

```
[             {EBCDIC        } ]
[CODE PROCESS {ASCII         } ]
[             {dbcs-code-name} ]
```

(Workstation) CODE defines the processing code system (EBCDIC or ASCII) CA-Easytrieve uses for working storage and for all files on which the CODE parameter was not specified on the FILE statement.

If not specified, the default is taken from the CA-Easytrieve/Workstation Site Options.

(Mainframe DBCS) Use CODE *dbcs-code-name* to define the DBCS code system to be used for all K and M fields for this file. If not specified here, the default is taken from the processing code system as defined in the CA-PSI Subsystems DBCS Options Table.

Note: Using multiple code systems in a program can result in a longer execution time due to code system conversions.

`[COMPILE]` COMPILE terminates execution after the completion of the syntax check and compile operations. Use COMPILE to review the code generated on the CA-Easytrieve Program Map (PMAP).

`[DEBUG]` DEBUG and its subparameters control generation of certain system outputs. These outputs are used to analyze programming errors that cause abnormal execution termination.

```
[CLIST  ]
[NOCLIST]
```

CLIST creates a condensed listing of the executable program produced by the compiler. NOCLIST inhibits this operation.

```
[PMAP  ]
[NOPMAP]
```

PMAP creates a complete listing of the executable program produced by the compiler. NOPMAP inhibits this operation.

CLIST and PMAP are mutually exclusive subparameters.

```
[DMAP  ]
[NODMAP]
```

DMAP creates a listing of the data map for each file and its associated fields. NODMAP inhibits this operation.

```
[FLDCHK  ]
[NOFLDCHK]
```

FLDCHK validates all data references during program execution. A data reference is invalid if a *field-name* was referenced in a file which had no active record. Invalid references (for example, data reference after end-of-file) might otherwise cause a program interruption or incorrect program results. NOFLDCHK inhibits this operation.

```
[FLOW  ]
[NOFLOW]
```

FLOW activates a trace of the statements being executed. The statement numbers are printed in the associated analysis report. NOFLOW inhibits this operation. FLOW and NOFLOW are ignored on the workstation.

```
[FLOWSIZ number-of-table-entries]
```

FLOWSIZ establishes the number of entries in the trace table for the flow option. *Number-of-table-entries* is a numeric value from 1 to 4096. FLOWSIZ is ignored on the workstation.

```
[STATE  ]
[NOSTATE]
```

STATE saves the statement number of the statement currently being executed. The statement number is then printed in the associated abnormal termination messages. NOSTATE inhibits this operation.

```
[XREF   {LONG } ]
[       {     } ]
[NOXREF {SHORT} ]
```

XREF causes the creation of a cross reference listing of each field name, file name, procedure name, screen name, report name, and statement label. LONG implies that entries are listed even though they are not referenced. SHORT causes only referenced entries to be listed. NOXREF inhibits this operation. XREF and NOXREF are ignored on the workstation.

```
[LINK (program-name [R])]
```

LINK terminates execution after the completion of syntax check and compile operations. On the mainframe, *program-name* is used to create the link edit control statement that names the program. In MVS systems, the optional subparameter R specifies that the program replaces an existing program with the same name. On the workstation, *program-name* names the file containing the object module (.OBJ).

Note: On the mainframe, if COMPILE, LINK, or SYNTAX is not specified, the object deck is generated without a NAME (MVS) or PHASE (VSE) card. For batch compilation, PARM LINK is not recommended because the *program-name* is generated from the batch compile JCL. In VSE systems, PARM LINK causes two PHASE cards to be generated for the object deck, which results in a non-executable load module. On the workstation, the object module file is named the same as the source program (program.OBJ).

```
[     {PARM  } ]
[LIST {      } ]
[     {NOPARM} ]
```

LIST controls the printing of certain system outputs.

PARM prints a compile summary and system parameters at the conclusion of the syntax check operation. NOPARM inhibits this operation.

```
[PLAN (planname [command-program-name])]
```

PLAN is an SQL parameter. Currently, it is used only by the mainframe DB2 SQL interface.

The PLAN parameter enables you to specify values for the "static-command-program" and its DB2 PLAN. The name you specify for the "static-command-program" must be a valid load module name. This name must be different from the *program-name* specified for the LINK parameter.

The value specified for *planname* must be the name of the DB2 PLAN that identifies the DBRM of the given "static-command-program." See the "SQL Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for information about how to generate the "static-command-program."

If not specified, *command-program-name* defaults to *planname.*

Because the link-edit of the "static-command-program" and the bind of the DB2 PLAN are performed outside the control of CA-Easytrieve, you must specify the correct names on the batch JCL to ensure successful execution of your program.

[PLANOPTS '*plan-options-module*'] PLANOPTS is an SQL parameter. Currently, this parameter is only used by the CA-Datacom/DB SQL interface.

Use PLANOPTS to specify the name of the plan options module that is to override the default CA-Pan/SQL plan options module, DQSMPLN@. See the *CA-Pan/SQL SQL Interface Installation Guide* for more information about generating a plan options module.

[PREPNAME (SQL-*access-module* ['*access-userid*'])] PREPNAME is an SQL parameter. Currently, it is used by the SQL/DS and CA-Datacom SQL interfaces.

For the SQL/DS SQL interface, the PREPNAME parameter enables you to specify the name of the access module or “package” that is to be associated with the SQL statements for this application program.

For the CA-Datacom SQL interface, PREPNAME enables you to specify the access plan.

For either database, The PREPNAME parameter also enables you to specify an owner ID (‘*access-userid*’) for the access module or access plan. See your specific database documentation for information about obtaining an authorization ID.

If PREPNAME is not specified, *SQL-access-module* defaults to *program-name* on the LINK parameter. If the LINK parameter is not specified, *SQL-access-module* defaults to the value specified in the Site Options Table.

Note: You should specify a unique value for the *SQL-access-module* for each CA-Easytrieve program. If you use the same name for either parameter value, database catalog contention can occur, or an existing access module could be replaced with another one. See your database administrator for information about establishing naming conventions.

PREPNAME can be abbreviated to PREP.

[SORT] SORT overrides the default parameters used to interface with your installation’s sort program. See the installation procedures on your product tape for details of these SORT parameters.

```
[       {NO                   } ]
[ALTSEQ {                     } ]
[       {(YES [alt-sort-table])} ]
```

ALTSEQ identifies the collating sequence table for the sort process. NO indicates usage of the standard table. YES identifies an alternate table. *Alt-sort-table* specifies the name of the table that you provide. When you omit *alt-sort-table*, the default name is EZTPAQTT.

[DEVICE *device-type*] DEVICE specifies the device type for dynamically allocated sort work data sets. *Device-type* can be any valid unit name or generic device type. DEVICE is ignored on the workstation.

```
[       {storage-amount          } ]
[MEMORY {                        } ]
[       {(MAX [-storage-released])} ]
```

MEMORY specifies the maximum amount of core storage used by the sort program. *Storage-amount* is the amount of storage made available for the sort and must be a value from 16 to 4096. MAX allows the sort program to obtain maximum storage available. *Storage-released* is the amount of storage released (for system use) after the MAX amount has been reserved. A minus sign must immediately precede *storage-released*. *Storage-amount* and *storage-released* values represent 1024-byte units of storage. MEMORY is ignored on the workstation.

```
[      {ALL      [CONSOLE]  }   ]
[      {         [PRINTER]  }   ]
[      {                    }   ]
[MSG ( {CRITICAL [CONSOLE]  } ) ]
[      {         [PRINTER]  }   ]
[      {DEFAULT             }   ]
[      {NO                  }   ]
```

Specify the level of messages to be output by the sort program.

ALL outputs all messages. CRITICAL outputs only critical level messages. DEFAULT outputs messages at the level specified when the sort program was installed. NO outputs no messages.

For ALL or CRITICAL messages, specify the location at which messages are received: PRINTER or CONSOLE.

MSG is ignored on the workstation.

[RELEASE *core-storage-amount*] RELEASE determines the amount of core storage reserved from the sort program. The value of *core-storage-amount* should be set large enough to supply all of the core storage needs of any exits used as a part of the sort process. *Core-storage-amount* must be a numeric value from 0 to 1024. The value represents 1024-byte units of storage.

RELEASE is ignored on the workstation.

[WORK *number-of-work-data-sets*])] WORK specifies the type and number of work data sets used by the sort.

The value of *number-of-work-data-sets* controls the allocation of work data sets. When *number-of-work-data-sets* is zero, you must supply DD statements for all work data sets (none are dynamically allocated). A *number-of-work-data-sets* value from 1 to 31 specifies the number of work data sets dynamically allocated by the sort program.

WORK is ignored on the workstation.

[SQLID '*auth-id*'] SQLID is an SQL parameter. Currently, this parameter is used only by the mainframe DB2 SQL interface.

SQLID enables you to change the authorization ID of your SQL session. If you specify a value for *'auth-id'*, the DB2 SET CURRENT SQLID command is executed by the DB2 SQL interface at compile time. For the SET CURRENT SQLID command to execute successfully, you must have installed the CA-Pan/SQL SQL Interface for a DB2 release of 2.1 or greater. You must also have the correct DB2 authorization to execute the SET CURRENT SQLID command. See your DB2 documentation for more information about the SET CURRENT SQLID command.

This parameter is in effect only for the compilation of your application program, unless your program is coded using automatic processing. If your program is coded using automatic processing, the SET CURRENT SQLID command is executed again at the start of runtime. For native SQL processing, you must code the SET CURRENT SQLID command in your program if you want to change the value for the current authorization ID.

See the "SQL Database Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

```
[          {FULL   } ]
[SQLSYNTAX {PARTIAL} ]
[          {NONE   } ]
```

Use SQLSYNTAX to specify the level of SQL syntax checking that is to be performed on the SQL statements coded in your program.

Specify FULL to indicate that detail level syntax checking should be performed. An SQL PREPARE statement is executed by the CA-Pan/SQL SQL Interface for those SQL statements that can be dynamically prepared. If you specify FULL, your DBMS catalog must be available to CA-Easytrieve.

Specify PARTIAL to indicate that SQL statements in your program should be syntax checked for valid commands and secondary keywords. No connection is made to the DBMS catalog unless you have coded the SQL INCLUDE statement. If you coded an SQL INCLUDE statement, your DBMS catalog must be available to CA-Easytrieve. Your program cannot be executed until it has been fully syntax checked, as described above.

Specify NONE with a BIND STATIC-ONLY parameter if you want syntax checking to be performed by the DB2 preprocessor in a batch environment. NONE causes partial syntax checking, as described above. If no compile errors are found, your program executes, unless CA-Easytrieve errors are found. No connection is made to the DBMS catalog unless you have coded the SQL INCLUDE statement. If you coded an SQL INCLUDE statement, your DBMS catalog must be available to CA-Easytrieve.

If you specify NONE for a non-DB2 environment, partial syntax checking is performed, but the program is not executed until full syntax checking is performed.

When running under the CA-Easytrieve interpreter, dynamic processing is always performed. An option of NONE is only effective for the batch compilation and execution of your program.

`[SSID '`*`ssid`*`']` SSID is an SQL parameter. Currently, SSID is used only by the DB2 , SYBASE, and CA-Ingres interfaces.

For mainframe DB2:

You can use SSID to specify the DB2 subsystem ID. If you specify this value, it is used at both compile and runtime. If you do not specify the DB2 subsystem ID, the subsystem ID from the Site Options Table is used.

If no DB2 subsystem ID is specified in the Site Options Table, the SQL interface uses the ID from the DB2 system default module DSNHDECP. The value of the subsystem ID is obtained at compile and runtime dynamically, therefore, there is no need to recompile your program to change the ID. See your DB2 systems programmer or administrator for the default values defined for your DB2 system.

For CA-Ingres, SYBASE, and UNIX DB2:

You can use SSID to specify the name of the database to which this session will connect. If you do not specify the subsystem ID, the subsystem ID from the Site Options Table is used. If no DB2 subsystem ID is specified in the site options table, DB2 uses the ID in the DB2DBDFT environment variable.

SSID is ignored in CICS and on the workstation.

`[SYNTAX]` SYNTAX terminates CA-Easytrieve processing after the syntax check operation.

`[TRANSID '`*`transid`*`']` (CICS only) Use TRANSID to specify the transaction identifier of a program to which control is transferred when the application user presses an attention key after a pseudo-conversational terminal I/O.

`[USERID ('`*`connect-userid`*`' ['`*`password`*`'])]` USERID is an SQL parameter. Currently, USERID is used by the SQL/DS, CA-IDMS, and UNIX SQL interfaces. USERID is used by the SQL interface to establish a connection to the database for compilation of the application program.

For SQL/DS:

You can use USERID to specify a valid userid and password for an explicit CONNECT.

For CA-IDMS:

You can use *'connect-userid'* to specify the CA-IDMS dictionary name for an explicit CONNECT. If you do not specify USERID, an implicit connection occurs according to the rules of the given database system.

For UNIX SQL interfaces:

You can use *'connect-userid'* to specify the user identifier under which this session will run. If you do not specify USERID, an explicit connection occurs without an *'identified-by'* clause. *'Password'* is ignored.

USERID is ignored on the workstation.

Note: USERID can be abbreviated as USER.

```
[                             [        {DISK  } ]    ]
[VFM ([buffer-core-storage] [DEVICE {        } ] ) ]
[                             [        {MEMORY} ]    ]
```

VFM establishes the work area parameters used by the CA-Easytrieve Virtual File Manager access method.

Buffer-core-storage specifies the amount of core storage made available for the buffer pool. Valid numeric values for *buffer-core-storage* are 6 to 4096. *Buffer-core-storage* represents 1024-byte units of storage.

DEVICE DISK reverts to disk device usage when the site option default is DEVICE MEMORY. DEVICE MEMORY inhibits the use of an overflow device.

VFM is ignored on the workstation.

```
[            {YES}              ]
[WORKFILE ( {    } [BLOCKMAX] )]
[            {NO }              ]
```

Use WORKFILE YES instead of VFM if you have multiple large reports in your program. NO is the default. In MVS systems that take advantage of the system-defined blocksize feature, use BLOCKMAX to automatically set the blocksize in your JCL to the largest possible blocksize for the track. See PRINT Statement Processing in the "Report Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

Usage Notes

Specification of the PARM statement is optional. Code the PARM statement only to modify the environment for your program. If used, the PARM statement must be the first statement in your CA-Easytrieve job.

```
Environment <============= PARM
  ...
Library
```

```
  ...
Activities
```

Code PARM statement parameters and their subparameters in any order. You must code multiple subparameters within parentheses.

PARM establishes program level parameters in the following areas:

- SYNTAX, COMPILE, and LINK determine the mode of execution.
- ABEXIT, DEBUG, and LIST establish control over system facilities associated with compiler output and execution error handling.
- VFM establishes system control parameters.
- SORT controls the interface to your installation's sort program.
- BIND, PLAN, PREPNAME, SQLID, SSID, USERID, PLANOPTS, and SQLSYNTAX establish parameters for SQL execution.
- TRANSID controls CICS execution.

See the *CA-Easytrieve/Online User Guide* for more information on controlling CA-Easytrieve with the PARM statement.

Examples

The following examples illustrate typical uses of the PARM statement:

```
PARM for production ...

     PARM LINK(MYPROG) DEBUG(CLIST, DMAP) +
         SORT (MSG (ALL, PRINTER))
     FILE PERSNL  FB(150 1800)
     %PERSNL
     JOB INPUT PERSNL NAME MYPROG
        PRINT REPORT1
     *
     REPORT REPORT1
     LINE EMPNAME DEPT

PARM for program testing ...

     PARM ABEXIT (SNAP) +
         DEBUG (PMAP, DMAP, FLDCHK, FLOW, +
         FLOWSIZ (20),  STATE)
     FILE PERSNL  FB(150 1800)
     %PERSNL
     JOB INPUT PERSNL NAME MYPROG
        PRINT REPORT1
     *
     REPORT REPORT1
     LINE EMPNAME DEPT SALARY-COD
```

PERFORM Statement

PERFORM transfers control to a procedure and, after the procedure has been executed, returns control to the next executable statement after the PERFORM statement.

Syntax

```
PERFORM proc-name
```

Parameters

proc-name Specify the name of the procedure to be executed.

Usage Notes

When CA-Easytrieve encounters the PERFORM statement, it immediately branches to the named procedure. After exiting from the procedure, program execution continues with the next executable statement following the just-executed PERFORM statement.

PERFORM statements in a procedure can invoke other procedures; this is called procedure nesting. However, recursion is not permitted. That is, procedure A can invoke procedure B, but procedure B cannot then invoke procedure A. If recursion is attempted, unpredictable results can occur.

Example

The following example illustrates the use of the PERFORM statement in executing a user procedure:

```
FILE PERSNL  FB(150 1800)
%PERSNL
XMAS-BONUS        W 4 P 2 VALUE 0
*
JOB INPUT PERSNL NAME MYPROG
   IF PAY-GROSS < 300.99
     PERFORM SPECIAL-BONUS
   ELSE
     PERFORM STANDARD-BONUS
   END-IF
   PRINT MYREPORT
*
SPECIAL-BONUS. PROC
   XMAS-BONUS = PAY-GROSS * 1.20
END-PROC
*
STANDARD-BONUS. PROC
   XMAS-BONUS = PAY-GROSS * 1.05
END-PROC
```

```
*
REPORT MYREPORT
LINE NAME-LAST XMAS-BONUS
```

POINT Statement

The POINT statement enables you to establish the position in an INDEXED or RELATIVE file from which subsequent data is sequentially retrieved. Data in the file becomes available only after the next successful sequential retrieval by either automatic file input or a GET statement.

Syntax

```
                           {= }
                           {EQ} {field-name}
POINT file-name [PRIOR] {GE} {          } [STATUS]
                           {GQ} {literal   }
                           {>=}
```

Parameters

file-name *File-name* must be the same as on a FILE statement that describes an INDEXED or RELATIVE file.

[PRIOR] Specify PRIOR if you want to use PRIOR on the GET statement. See the GET Statement for more information.

```
{= }
{EQ}
{GE}
{GQ}
{>=}
```

Equal operators (= and EQ) initiate a file position search, based on an exact match between the file's keys and the search value. The greater-than operators (GE, GQ, and >=) initiate a file position search, based on a file's key being equal to or greater than the search value.

```
{field-name}
{          }
{literal   }
```

The search value can be any valid *field-name* or *literal*. Alphanumeric literals must be enclosed within single quotes. CA-Easytrieve does not change the data format of *field-name* or *literal*. The search value must have the same length as the file's key. RELATIVE files require *field-name* to be a 4-byte binary integer field. *Field-name* cannot be nullable. *Literal* is not allowed for a RELATIVE file.

[STATUS] Specify the STATUS parameter whenever the possibility exists for an unsatisfactory completion of the input/output request..

STATUS checks input/output processing to see if it was performed correctly. STATUS causes the file's FILE-STATUS field to be set with the appropriate return code. Refer to Appendix A, "System-Defined Fields," to determine the meaning of the contents of FILE-STATUS. Normally, a zero or non-zero test is sufficient.

Note: FILE-STATUS is not defined if you do not specify a file type parameter on the FILE statement.

If you do not code STATUS and the operating system returns a non-zero status, CA-Easytrieve issues an appropriate diagnostic message.

In addition to FILE-STATUS, IF EOF *file-name* is true when the search value is greater than the highest key in the file.

Note: CICS does not set EOF.

Usage Notes

You cannot use a file presence test (IF *file-name*) to test the success of a POINT.

You cannot issue a GET PRIOR statement following a POINT statement, or a GET statement following a POINT PRIOR statement. See the GET Statement for more information.

GE is not supported for POINT PRIOR when the underlying access method does not support it.

Example

The following example illustrates the use of POINT:

```
FILE PERSNL INDEXED
%PERSNL
JOB INPUT NULL NAME MYPROG
   POINT PERSNL GE '01963' STATUS
   IF FILE-STATUS NE 0 OR EOF PERSNL
      DISPLAY 'BAD POINT...FILE STATUS= ' FILE-STATUS
      STOP
   END-IF
   GET PERSNL STATUS
   IF FILE-STATUS NE 0
     DISPLAY 'BAD GET...FILE STATUS= ' FILE-STATUS
   ELSE
     DISPLAY HEX PERSNL
   END-IF
   STOP
```

See File Processing in the *CA-Easytrieve Programmer Guide* for more detailed examples on the use of POINT in file processing.

POP Statement

POP is a listing control statement that restores previous listing control indicators.

Syntax

```
POP
```

Usage Notes

POP is especially useful in macros to control the listing of the macro expansion without affecting listing control indicators outside the macro. Use the PUSH statement to save the current indicators. You can then set listing control indicators for use during macro expansion. The POP statement restores the saved indicators.

Note: Use the LIST statement to set listing control indicators.

You can place POP statements anywhere in the CA-Easytrieve source code. POP must be on a record by itself. POP does not appear in the printed output.

PRINT Statement

The PRINT statement produces report output. Issue the PRINT statement to initiate a printed line.

Syntax

```
PRINT [report-name]
```

Parameters

[*report-name*] *Report-name* is the name of the report as specified on a REPORT statement. If not given, it is assumed to be the first report in the JOB activity.

Usage Notes

In general, report output is not written directly to a report's printer file as with DISPLAY, but is scheduled for deferred formatting and writing to the report's printer file, perhaps following re-sequencing of an intermediate file.

See Report Processing in the *CA-Easytrieve Programmer Guide* for detailed examples on the use of PRINT in report processing.

When you require an intermediate file (referred to as a report work file) for a report, executing PRINT causes fixed format records (called spool records) to be output to the work file. CA-Easytrieve determines the format of these records, which includes all the fields required to produce the report except those in S type working storage.

Example

```
FILE PERSNL  FB(150 1800)
%PERSNL
JOB INPUT PERSNL NAME PRINT-RPT
  PRINT REPORT1
  REPORT REPORT1
    TITLE 'PERSONNEL REPORT'
    LINE EMP# SSN EMPNAME
```

PROC Statement

The PROC statement is used to begin a CA-Easytrieve procedure. A procedure is a group of user-written CA-Easytrieve statements designed to accomplish a particular objective. The syntax of a procedure has two statements:

- A label naming the procedure
- The PROC statement.

Syntax

Format 1

```
proc-name. PROC
   statement-1
   ...
   statement-n
END-PROC
```

Format 2

```
proc-name
PROC
   statement-1
   ...
   statement-n
END-PROC
```

Parameters

proc-name *Proc-name* is a label that identifies the procedure. A label can:

- Be up to 128 characters in length

- Contain any character other than a delimiter
- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

Proc-name must be followed by the keyword PROC as a separate statement.

`statement-1...n` *Statement-1* through *statement-n* are the statements that accomplish your procedure's task.

`END-PROC` The END-PROC statement delimits the statements contained in the procedure.

Usage Notes

In most cases, you can code any statement in a procedure. However, you cannot code certain input/output statements (such as GET, PUT) in procedures invoked during SORT or REPORT processing.

PERFORM statements within a procedure can invoke other procedures; this is called procedure nesting. However, recursion is not permitted. That is, procedure A can invoke procedure B, but procedure B cannot then invoke procedure A. If recursion is attempted, unpredictable results can occur.

Screens and reports can contain special-named procedures. Each procedure is explained separately in this manual.

Screen Procedures	Report Procedures
AFTER-SCREEN	AFTER-BREAK
BEFORE-SCREEN	AFTER-LINE
INITIATION	BEFORE-BREAK
TERMINATION	BEFORE-LINE
	ENDPAGE
	REPORT-INPUT
	TERMINATION

PROGRAM Statement

The PROGRAM statement identifies and initiates a processing activity that can optionally initiate JOB, SORT, and SCREEN activities.

A PROGRAM statement is required when:

- A program is the target of a TRANSFER from another program and parameters are passed between programs. See the TRANSFER Statement.
- A program is the child program LINKed to from a parent program and parameters are passed between programs. See the LINK Statement.
- A parameter is passed to a program invoked from the operating system.
- You want to selectively execute other activities or execute a single activity multiple times.

Syntax

```
                           [        [ACTIVITY  ] [TERMINAL  ] ]
PROGRAM [NAME program-name] [COMMIT ([          ] [          ])] +
                           [        [NOACTIVITY] [NOTERMINAL] ]

        [USING field-name] [GIVING field-name]
```

Parameters

[NAME *program-name*] The NAME parameter names the processing activity. *Program-name* identifies the program. *Program-name* can:

- Be up to 128 characters in length
- Contain any character other than a delimiter
- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

This parameter is used for documentation purposes.

```
        [ACTIVITY  ] [TERMINAL  ] ]
COMMIT ([          ] [          ])]
        [NOACTIVITY] [NOTERMINAL] ]
```

Specify the COMMIT parameter to control the logical unit of work. COMMIT indicates when the activity commits recoverable work. Each commit point posts all updates, additions and deletions, terminates holds, and closes SQL cursors.

Specify ACTIVITY to commit all recoverable work during the normal termination of the activity. Specify NOACTIVITY to tell CA-Easytrieve not to commit at the end of the activity. ACTIVITY is the default.

Specify TERMINAL to commit all recoverable work during any terminal I/O operation. In CICS, this results in terminal I/O being performed in a pseudo-conversational mode. Specify NOTERMINAL to tell CA-Easytrieve not to commit during a terminal I/O. TERMINAL is the default.

If this program is linked to by another CA-Easytrieve program, this program performs terminal I/O as if NOTERMINAL was specified.

Note: You can also issue your own COMMIT and ROLLBACK statements to commit or recover work on a controlled basis.

See the *CA-Easytrieve Programmer Guide* for more information about commit processing.

`[USING` ***`field-name`*** **`]`** Specify USING to indicate that this program (child program) can accept a parameter from the parent program or operating system. *Field-name* is the name of a field to which the parameter is passed.

`[GIVING` ***`field-name`*** **`]`** Code GIVING to return a single parameter to the parent program. *Field-name* is the name of a field containing the parameter you want to return to the parent program.

Usage Notes

A PROGRAM statement defines an activity in which JOB, SORT, and SCREEN activities can be conditionally invoked. If a PROGRAM statement is not present, any JOB or SORT activities are sequentially executed until a SCREEN activity is detected. The SCREEN activity is then executed. The sequential execution of activities does not proceed past the first SCREEN activity. Any remaining activities must be executed by the first SCREEN activity.

A program terminates when:

- The bottom of the activity is reached
- A STOP EXECUTE statement is executed
- A TRANSFER statement is executed.

A PROGRAM statement can be used to execute a sequence of statements. You can also use a JOB INPUT NULL statement, however, you must then execute a STOP statement to terminate the activity.

Example

```
FILE ...
  ...
PROGRAM NAME EXAMPLE1
  EXECUTE JOB1
  EXECUTE PANEL1
JOB NAME JOB1
  ...
SCREEN NAME PANEL1
  ...

is equivalent to:
```

```
FILE ...
  ...
JOB NAME JOB1
  ...
SCREEN NAME PANEL1
  ...
```

PUSH Statement

PUSH is a listing control statement that saves the current listing control indicators.

Syntax

```
PUSH
```

Usage Notes

PUSH is useful in macros to control the listing of the macro expansion without affecting the listing control indicators outside the macro. PUSH saves the current indicators. You can then set the listing control indicators for use during macro expansion. Use the POP statement to restore the saved indicators.

Note: Use the LIST statement to set listing control indicators.

You can code a PUSH statement anywhere in CA-Easytrieve source code. PUSH must be on a record by itself. PUSH does not appear in the printed output.

PUT Statement

The PUT statement performs sequential file output. PUT outputs records to SEQUENTIAL files and also adds consecutive records (mass sequential insertion) to an INDEXED or RELATIVE file.

Syntax

```
                     [     {input-file-name]  }]
PUT output-file-name [FROM {                  }][STATUS]
                     [     {input-record-name}]
```

Parameters

output-file-name The *output-file-name* parameter identifies the output file.

```
[     {input-file-name  }]
[FROM {                 }]
[     {input-record-name}]
```

FROM identifies the file or record from which the current record is copied.

When *input-file-name* is specified, the length of the output data is the same as *output-file-name:*RECORD-LENGTH. The current value of *input-file-name:*RECORD-LENGTH is equal to the length of the input data. However, if the length of the output file is greater than the length of the input file, the excess storage is not initialized. Also, use of the FROM parameter does not update the data area of the output file.

For more information about RECORD-LENGTH, see File Processing in the *CA-Easytrieve Programmer Guide*.

[STATUS] Specify the STATUS parameter whenever the possibility exists for an unsatisfactory completion of the input/output request.

STATUS checks input/output processing to see if it was performed properly. STATUS causes the file's FILE-STATUS field to be set with the appropriate return code. See Appendix A, "System-Defined Fields," to determine the meaning of the contents of FILE-STATUS. Normally, a zero or non-zero test is sufficient.

Note: FILE-STATUS is not defined if you do not specify a file type parameter on the FILE statement.

If you do not code STATUS and the operating system returns a non-zero status, CA-Easytrieve issues an appropriate diagnostic message.

Usage Notes

To take advantage of VSAM's mass-sequential-insertion capabilities, you can use the PUT statement to add many records to the same place in any established VSAM file.

If you use the PUT statement, you must include the UPDATE parameter on the FILE statement for RELATIVE or INDEXED files. You must specify CREATE for SEQUENTIAL files. UPDATE informs CA-Easytrieve that all input records can potentially be updated or deleted.

Example

```
FILE FILEA INDEXED UPDATE
%PERSNL
FILE PERSNL
COPY FILEA
JOB INPUT PERSNL NAME MYPROG
  PUT FILEA FROM PERSNL STATUS
  IF FILE-STATUS NE 0
```

```
    DISPLAY 'ADD FAILED'
    DISPLAY HEX PERSNL
    STOP
  END-IF
```

READ Statement

READ provides random access to INDEXED and RELATIVE files.

Syntax

```
                   {key-field-name} [HOLD  ]
READ file-name KEY {               } [      ] [STATUS]
                   {'key-literal' } [NOHOLD]
```

Parameters

`file-name` Specify the *file-name* of the INDEXED or RELATIVE file to be randomly accessed.

```
    {key-field-name}
KEY {               }
    {'key-literal' }
```

You must provide the key to the desired record. CA-Easytrieve uses the contents of *key-field-name* or *'key-literal'* in a search for a corresponding record in the file. Alphanumeric literals must be enclosed within single quotes. CA-Easytrieve does not change the data format of *key-field-name* or *'key-literal.'* The access method can require that the search value have the same length as the file's key. *Key-field-name* cannot be nullable.

RELATIVE files require *key-field-name* to be a 4-byte binary integer field. *'Key-literal'* is not allowed for a RELATIVE file.

```
[HOLD  ]
[NOHOLD]
```

CA-Easytrieve automatically issues a hold request for records when UPDATE is specified on the FILE statement. You use NOHOLD to override this process.

Specify HOLD to hold a record for update. This is the default when UPDATE is specified for the file. HOLD is invalid if UPDATE is not specified on the FILE statement. HOLD does not mean you are required to perform the update. It holds the position of the file and can also lock the record (CICS and workstation LANs). Records are automatically released when the update operation completes or a commit point is taken. You can also manually release the hold on any record with the RELEASE statement.

NOHOLD specifies that a record is not held for update.

`[STATUS]` Specify the STATUS parameter whenever the possibility exists for an unsatisfactory completion of the input/output request.

STATUS checks input/output processing to see if it was performed properly. STATUS causes the file's FILE-STATUS field to be set with the appropriate return code. See Appendix A, "System-Defined Fields," to determine the meaning of the contents of FILE-STATUS. Normally, a zero or non-zero test is sufficient.

Note: FILE-STATUS is not defined if you do not specify a file type parameter on the FILE statement.

If you do not code STATUS and the operating system returns a non-zero status, CA-Easytrieve issues an appropriate diagnostic message.

Usage Notes

The key specified is normally a working storage field or a field in another file. It cannot be the file's key field unless WORKAREA is specified to make the field available prior to the READ.

You can use a file presence test (IF *file-name*) to determine the success of the READ. The test is true when the last GET or READ was successful.

Example

```
FILE PERSNL INDEXED
%PERSNL
FILE INKEYS SEQUENTIAL
  WHO 1 5 N
  TOTAL-NET W 5 P 2
JOB INPUT INKEYS NAME MYPROG FINISH DISPLAY-TOTAL
   READ PERSNL KEY WHO STATUS
   IF PERSNL:FILE-STATUS NE 0
      DISPLAY 'UNSUCCESSFUL READ +
         PERFORMED ON FILE PERSNL'  +
         +2 'KEY= ' WHO
   ELSE
      TOTAL-NET = TOTAL-NET + PAY-NET
   END-IF
   DISPLAY-TOTAL. PROC
     DISPLAY 'TOTAL NET PAY' TOTAL-NET
   END-PROC
```

RECORD Statement (CA-IDMS and IMS/DLI)

Code RECORD statements (Format 1) following the FILE statement to identify the CA-IDMS database records available for automatic or controlled processing.

RECORD statements (Format 2) identify the IMS/DLI database segments that are to be available for processing.

Syntax

Format 1 (CA-IDMS)

```
RECORD record-name record-length [KEY (field-name ...)]
```

Format 2 (IMS/DLI)

```
RECORD segment-name segment-length [parent-segment-name]  +
         [KEY (key-field-name, key-field-location, key-field-length)]
```

Parameters

Format 1 (CA-IDMS)

record-name *Record-name* is the one to sixteen-character name of the record as defined in the subschema.

record-length *Record-length* is a positive integer that designates the length of the record as defined in the subschema.

[KEY (*field-name* ...)] KEY is an optional parameter that identifies the CALC key(s) of the record. The KEY parameter is required only for the root record when using the tickler file. *Field-name* is the one to thirty-character name used to designate one of the record's CALC keys. *Field-name* must correspond to a CALC key field defined with the DDL for the record. You must code the *field-name* parameter for each CALC key defined for the record.

In addition, a DEFINE statement for each *field-name* must be coded following the RECORD statement. The DEFINE statement can be intermingled with DEFINE statements for other non-key fields of the record.

Format 2 (IMS/DLI)

segment-name *Segment-name* is the one to eight-character name of the segment. This name must correspond to the name of a segment in the DBD.

segment-length *Segment-length* is a positive numeric integer that designates the length of the segment.

[*parent-segment-name*] *Parent-segment-name* is an optional parameter which designates the parent of *segment-name*. This parameter is not coded for the root segment, but it is required for all other segments.

[KEY] The optional KEY parameter identifies the sequence field for the segment.

The KEY parameter is not necessary for the RECORD statement that defines the lowest segment in a path. The KEY parameter is required for the root segment when using the tickler file.

(*key-field-name*) *Key-field-name* is the one to eight-character name used to designate the keyfield to the IMS/DLI database. The name must correspond to the sequence field named in the DBD.

(*key-field-location*) *Key-field-location* is a positive numeric integer that specifies the location of the keyfield within the segment.

(*key-field-length*) *Key-field-length* is a positive numeric integer that specifies the length of the keyfield.

Usage Notes

CA-IDMS

All fields defined following the RECORD statement are part of this record. The name of each field must be unique within the record. However, the field does not have to be unique within the file that contains the record being defined. If a field defined in another record has the same name as a field defined in this record, then all references to either field must be qualified with the name of the field's containing record.

Note: The RECORD statement cannot be used to define logical records. To define a logical record, use the LOGICAL-RECORD statement. To define an element record within a logical record, use the ELEMENT-RECORD statement.

IMS/DLI

RECORD allocates a work space which contains the segment data during execution. Field definition statements, coded immediately following a RECORD statement, relate to data fields within that segment. One RECORD statement must be coded for each segment of the database to be processed. They must be coded in the same order as in the PSB which defines the database. All segments of a database need not be defined, but the parent segment of each RECORD must be coded because incomplete paths are not supported.

REFRESH Statement

The REFRESH statement is used in the AFTER-SCREEN procedure to restore the initial screen image by rebuilding it with the current values of the program fields.

Syntax

```
REFRESH
```

Usage Notes

When used as the result of pressing an IMMEDIATE key, REFRESH re-displays the screen image with the original data displayed on the screen. This is useful when the terminal user enters erroneous data on the screen and wants to restore the screen with its original data.

When used as the result of a non-IMMEDIATE key, REFRESH can be used to rebuild the screen using current data from the screen.

REFRESH can also be invoked directly by pressing a particular attention key. See the KEY Statement for more information.

Example

The following example illustrates the REFRESH statement being invoked when F6 is pressed. Because F6 is not an IMMEDIATE key, the current values of QTY and PRICE are used to compute the extended price. F5 is used to clear erroneous data from the screen.

```
DEFINE EXT-PRICE W 4 P 2
DEFINE QTY W 4 P 0
DEFINE PRICE W 4 P 2
SCREEN NAME TEST-REFRESH
  KEY F2 NAME 'Reset to zero'
  KEY F3 EXIT NAME 'Exit'
  KEY F5 IMMEDIATE REFRESH NAME 'Refresh screen'
  KEY F6           NAME 'Compute Ext Price'
  TITLE 'TEST REFRESH'
  ROW 3 'Quantity . .' QTY
  ROW 5 'Price  . . .' PRICE
  ROW 7 'Ext Price  .' EXT-PRICE
  BEFORE SCREEN. PROC
    MOVE ZERO TO QTY, PRICE, EXT-PRICE
  END-PROC
  AFTER-SCREEN. PROC
    IF KEY-PRESSED = F6
      EXT-PRICE = QTY * PRICE
      REFRESH
    END-IF
  END-PROC
```

RELEASE Statement

The RELEASE statement explicitly releases the hold on any record in a file.

Syntax

```
RELEASE file-name
```

Parameters

file-name *File-name* is the name of a file.

Usage Notes

CA-Easytrieve automatically issues a hold request for GETs and READs when UPDATE is specified on the FILE statement. Records are automatically released when the update operation completes or a commit point is taken. Alternatively, you can use the RELEASE statement to manually release the hold on a record. If HOLD is not specified, RELEASE is ignored.

Example

```
READ PERSNL KEY '01193' HOLD
...
IF ...
  WRITE PERSNL UPDATE
ELSE
  RELEASE PERSNL
END-IF
```

REPEAT and END-REPEAT Statements

The REPEAT/END-REPEAT construct is used to display arrays on a screen.

Syntax

```
                 [                  [     {start-field-name} ] ]
REPEAT number [TIMES] [VARYING subscript [FROM {                  } ] ]   +
                 [                  [     {start-integer   } ] ]

       [ROW row-number]

ROW statements

END-REPEAT
```

Parameters

number *Number* is the number of times the group of ROW statements in the REPEAT construct is repeated.

[TIMES] Optionally, code TIMES for statement readability.

[VARYING *subscript*] VARYING is an optional parameter that causes CA-Easytrieve to automatically increment a subscript field (*subscript*).

The *subscript* is incremented by 1 the number of times specified by *number*.

Subscript can be the name of a previously defined field. However, if not defined, CA-Easytrieve automatically defines the field as a 2 B 0 field. If you defined the field, you must have defined it as numeric (N, P, U, B, I) with zero or no decimal places.

```
[     {start-field-name} ]
[FROM {                } ]
[     {start-integer   } ]
```

FROM is an optional parameter that defines the initial value for the REPEAT subscript. *Subscript* is automatically incremented by 1 from either *start-field-name* or *start-integer* for each iteration of the group of ROW statements. If FROM is omitted, the subscript starts at 1.

[ROW *row-number*] Specify the *row-number* on which the REPEATing group of rows starts. If not specified, the REPEATing group of rows starts on the last screen row specified plus one.

ROW *statements* Code one or more ROW statements to be repeated. ROW statements are coded in a REPEAT/END-REPEAT construct. See the ROW Statement.

Note: ROW statements to be repeated cannot specify explicit *row-number*s.

END-REPEAT END-REPEAT terminates the body of the REPEAT statement. END-REPEAT must be specified after each REPEAT statement and its associated ROW statements.

Usage Notes

Each array field on a ROW statement can be subscripted by the *subscript* specified in the VARYING parameter. Optionally, array fields on ROW statements can contain a subscript for a second dimension. However, CA-Easytrieve does not automatically increment this second subscript.

Example

The following example illustrates how to display an array on a screen. The REPEAT construct displays both a one-dimensional array (WS-NAME), and a two-dimensional array (WS-STAT). Starting at row 4, CA-Easytrieve displays the first occurrence of the fields. The second dimension of WS-STAT is stated explicitly. CA-Easytrieve increments USER-SUB and displays the next occurrence until 15 occurrences are displayed.

This code:

```
...
WS-EMPLOYEE W 33 A OCCURS 30        . * 2-DIMENSIONAL TABLE OF
  WS-NAME     WS-EMPLOYEE      30 A . * 30 EMPLOYEES CONTAINING:
  WS-STATUSES WS-EMPLOYEE +30   3 A . * EMPLOYEE NAME AND
    WS-STAT WS-STATUSES 1 A OCCURS 3. * 3 STATUSES
...
SCREEN NAME EMPLOYEE-LIST
  TITLE 'List of Employees'
  ROW 3 'Name' COL 30 'Statuses'
  REPEAT 15 TIMES VARYING USER-SUB
     ROW WS-NAME (USER-SUB) +
         WS-STAT (USER-SUB, 1) WS-STAT (USER-SUB, 2) WS-STAT (USER-SUB, 3)
  END-REPEAT
```

Produces:

```
                                  List of Employees

Name                          Statuses
WIMN, GLORIA                  F G O
BERG, NANCY                   C
CORNING, GEORGE               I T
...
```

REPORT Statement

The REPORT statement allows you to define the type and characteristics of a report. Although you can specify a large number of REPORT statement parameters, you will probably produce most reports using default parameter values specified in the Site Options Table.

REPORT statement parameters fall into five basic groups:

- Format determination parameters
- Label parameters
- File directing parameters
- Spacing control parameters
- Testing aid parameters.

Syntax

```
REPORT [report-name] +

[SUMMARY] +                                                }
[SUMFILE summary-file-name] +                              }
[SUMSPACE sumfield-addition] +                             }
[TALLYSIZE tally-print-size] +                             }
                                                           }
[       {EVERY} ]                                          }
[DTLCTL {FIRST} ] +                                        } Format
[       {NONE } ]                                          } Determination
                                                           } Parameters
[       {   [ALL ]               } ]                       }
[SUMCTL { ( [HIAR] [DTLCOPY   ] ) } ] +                    }
[       {   [NONE] [DTLCOPYALL]   } ]                      }
[       {   [TAG ]               } ]                       }
                                                           }

[         [ACROSS number-of-labels]   ]                    }
[LABELS ( [DOWN number-of-lines   ] ) ] +                  } Label
[         [SIZE label-length      ]   ]                    } Parameters
[         [NEWPAGE                ]   ]                    }

                                                           }
[FILE work-file-name] +                                    } File Directing
[PRINTER receive-file-name] +                              } Parameters

[PAGESIZE (line-page-size [display-page-size])] +          }
[LINESIZE line-length] +                                   }
[SKIP number-of-lines] +                                   }
[SPACE number-of-spaces] +                                 }
[TITLESKIP number-of-lines] +                              }
[CONTROLSKIP number-of-lines] +                            }
                                                           }
[SPREAD  ] +                                               }
[NOSPREAD]                                                 } Spacing Control
                                                           } Parameters
[NOADJUST] +                                               }
                                                           }
[NODATE   ]                                                }
[LONGDATE ] +                                              }
[SHORTDATE]                                                }
                                                           }
[NOPAGE] +                                                 }
[NOHEADING] +                                              }

                                                           }
[LIMIT number-of-records] +                                } Testing Aid
[EVERY n-number-of-lines]                                  } Parameters
```

Format Determination Parameters

[*report-name*] *Report-name* identifies the report. It is optional when there is only one report in a JOB activity. If you code multiple reports, the first report can be unnamed but all others must be named. Each *report-name* must be unique in the JOB activity. At least one *report-name* must be coded on a PRINT *report-name* statement. For unnamed reports, code the PRINT statement without a *report-name* parameter.

Report-name can:

- Be up to 128 characters in length
- Contain any character other than a delimiter

- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

[SUMMARY] On control reports, SUMMARY inhibits printing of detail data. Only control totals are printed.

[SUMFILE *summary-file-name*] Optionally, use SUMFILE to generate a summary file that contains the control and summary field values. *Summary-file-name* identifies the file to contain the summary data.

[SUMSPACE *sumfield-addition*] Use SUMSPACE to define the print size for total fields on a control report. *Sumfield-addition* is added to the length (in digits) of the field to define its print size. This expansion is necessary to prevent the loss of significant data due to overflow conditions. The resulting print length is limited to a maximum of 18 digits. Valid values for *sumfield-addition* are 0 through 9. No additional numeric edit characters are included in the resulting edit mask. For example, totals such as 55555,555.55 can appear.

[TALLYSIZE *tally-print-size*] Use TALLYSIZE to set the print size for the field TALLY. Valid values for *tally-print-size* are 1 through 18. The number of digits used for TALLY on a summary line are the sum of the values of TALLYSIZE and SUMSPACE.

```
[       {EVERY} ]
[DTLCTL {FIRST} ]
[       {NONE } ]
```

DTLCTL optionally defines detail line printing characteristics.

Specify EVERY to print the value of all control fields on every detail line.

Specify FIRST to print the value of all control fields on the first detail line of a page and on the first detail line after each control break. Control field values are not printed on all other detail lines.

Specify NONE to inhibit the printing of control field values on detail lines.

```
[       {   [ALL ]                } ]
[SUMCTL { ( [HIAR] [DTLCOPY   ] )  } ] +
[       {   [NONE] [DTLCOPYALL]    } ]
[       {   [TAG ]                } ]
```

SUMCTL optionally defines total line printing characteristics.

Specify ALL to print control field values on every total line.

Specify HIAR to print control field values in a hierarchical fashion on total lines. Only values for control fields on the same hierarchical level, or higher than the breaking control field, are printed on the associated total line.

Specify NONE to inhibit printing of control field values on total lines.

Specify TAG to print *control-field-name* TOTAL as an annotation to the left of the associated total line where *control-field-name* is the *field-name* for the breaking control field. There must be sufficient unused space on the left side of the total line for this annotation.

Specify DTLCOPY to print detail information on total lines. Normally, only control field values and associated totals are printed on total lines. Coding DTLCOPY prints the detail field contents, prior to the break, on the total line. These fields are printed only when LEVEL is one (1).

Specify DTLCOPYALL to print detail fields for all control breaks.

Label Parameters

```
[         [ACROSS number-of-labels]   ]
[LABELS ( [DOWN number-of-lines   ] ) ] +
[         [SIZE label-length      ]   ]
[         [NEWPAGE                ]   ]
```

Specify LABELS to indicate that the report is a label report.

Note: The NOHEADING and NOADJUST options are automatically activated when you specify LABELS; therefore, you cannot specify TITLE and HEADING statements. You cannot use LABELS with SUMMARY.

Specify ACROSS *number-of-labels* to define the number of labels printed across the print line.

Specify DOWN to define the number of lines in a label. The value of *number-of-lines* is the number of lines reserved for each label. The value range for *number-of-lines* is 1 through ‘nnn,’ where ‘nnn’ is at least as large as the largest corresponding ‘LINE nnn’ value.

Specify SIZE to set the length of each label. The value of *label-length* is the number of print positions on a label. *Label-length* has a value range from 1 to ‘nnn,’ where ‘nnn’ is the length of the label.

Note: LABELS cannot be specified for extended reporting printers.

NEWPAGE controls the printing of the first line (LINE 01) of each label. When coded, NEWPAGE associates a printer top-of-form request with the first line of each label.

The following algorithm confines the overall size of labels:

```
LINESIZE >= (ACROSS - 1) * SIZE + (number of print positions on an
                                   individual label)
```

File Directing Parameters

`[FILE` *`work-file-name`*`]` Optionally, specify FILE to identify the work file used for very large reports. Code this parameter when the default VFM work file is too small to contain the report data. *Work-file-name* identifies the FILE that receives the work file data.

Note: You should not use the FILE parameter in CICS. An execution error occurs when *work-file-name* is not a virtual file.

Note: Instead of coding the FILE parameter for each report in your program, you can use the WORKFILE YES parameter on the PARM statement. See PARM Statement earlier in this chapter, and PRINT Statement Processing in the "Report Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

`[PRINTER` *`receive-file-name`*`]` Optionally, specify PRINTER to direct the report's printed output. *Receive-file-name* identifies the FILE that receives the report. This file must have the PRINTER or EXTENDED attribute specified. The default is the CA-Easytrieve standard print output file: SYSPRINT. The actual destination of SYSPRINT is determined by a site option. See your system administrator for more information. See the *CA-Easytrieve Programmer Guide* for more information about PRINTER files.

If the system print output file or *receive-file-name* has been associated with an extended reporting printer, then CA-Easytrieve automatically formats the report to satisfy the requirements defined for that extended reporting printer. CA-Easytrieve restricts the support of extended reporting facilities to those reports that are output to printer files that have been associated with an extended reporting printer.

Spacing Control Parameters

Each of the following parameters modifies the default spacing of a report page. You normally do not use these parameters; however, they are available to support unique report spacing requirements.

`[PAGESIZE (`*`line-page-size`*` [`*`display-page-size`*`])]` Specify PAGESIZE to define the logical print length of a printed page. *Line-page-size* must be an unsigned integer from 1 to 32767, and must be at least as large as the sum of:

- *Title-number* of the last TITLE statement
- *Number-of-lines* of TITLESKIP
- Number of HEADING lines plus one
- *Line-number* of the last LINE statement
- *Number-of-lines* of SKIP.

In other words, at least one line group must fit on a report page.

Specify an asterisk (*) for *line-page-size* if you want to change *display-page-size* without changing the default *line-page-size*.

Display-page-size must be zero or greater than or equal to the *line-page-size*.

When CA-Easytrieve processes a LINE statement, it compares the line count to *line-page-size*. If the line count is less than *line-page-size*, the LINE statement performs the BEFORE-LINE procedure and then prints the line. If the line count is greater than or equal to *line-page-size*, the LINE statement performs the ENDPAGE procedure, processes TITLEs, performs the BEFORE-LINE procedure, and finally prints the line. The line count is not compared again to *line-page-size* after the LINE statement performs the BEFORE-LINE procedure.

Specify a value greater than zero for *display-page-size* to allow the DISPLAY statement to generate page breaks. When *display-page-size* is greater than zero, the line count is compared to *display-page-size*. If the line count is greater than *display-page-size* then the DISPLAY statement performs the ENDPAGE procedure and generates a page break with TITLEs.

Specify zero for *display-page-size* to inhibit DISPLAY statement generated page breaks. When *display-page-size* is zero, the DISPLAY statement does not compare line count to *display-page-size*, and a page break is not generated.

The DISPLAY statement always increases line count, regardless of the *display-page-size* value.

When the report is directed to an extended reporting printer that does not support a Forms Control Block (FCB), then CA-Easytrieve multiplies *line-page-size* by the default height of the assigned extended reporting printer. This enables CA-Easytrieve to compare PAGESIZE with the heights of fonts used on the report as they are both in the same base unit (the H-unit). The value of *line-page-size* multiplied by the default height of the assigned extended reporting printer cannot exceed the maximum page length of that extended reporting printer.

`[LINESIZE` ***`line-length`*** **`]`** Code the LINESIZE parameter to specify the maximum number of data characters that can be printed on a line. *Line-length* must be an unsigned integer from 1 to 32767.

Line-length must be at least one less than the length of the data portion of the file's logical record. If the FILE definition does not provide the file's format and logical record length, then no compile time verification of the *line-length* is done.

The default value of LINESIZE is calculated as one less than the data portion of the logical record if the file format and record length are known at compile time. Otherwise, the default is taken from the LINESIZE site option.

There are additional control characters (forms control information) that also must be stored in a logical record. If one of the record format parameters is specified, it must be large enough to hold both the forms control information and the data characters. The value of *line-length* must be less than or equal to the maximum record length minus the size of the forms control information.

The first character in a PRINTER file contains the ASA carriage control information.

When the report is assigned to an extended reporting printer that is not a standard line printer, the maximum value of *line-length* is not dependent upon the record size of the print data set. The insertion of Overprint and Function Codes into print records plus the support of different fonts on the same print line all impact the relationship between LINESIZE and print data set record size. CA-Easytrieve supports any LINESIZE provided *line-length* multiplied by the value of the assigned extended reporting printers default width does not exceed the maximum page width of that extended reporting printer.

Line-length overrides the value defined in the Site Options Table. If the report is directed to an extended reporting printer, CA-Easytrieve multiplies *line-length* by the default width of the assigned extended reporting printer. This value defines the width of the print line in terms of the extended reporting printer's W-unit.

`[SKIP` ***`number-of-lines`*****`]`** Specify SKIP to define the number of blank lines to be inserted between line groups (between the last 'LINE nnn' and the next LINE 01). *Number-of-lines* has a valid range of 0 to 'nnn,' where 'nnn' allows for the printing of at least one line group on each page. When you specify a value of 0, a line group containing multiple lines can be spanned across a page break. A non-zero value inhibits this spanning.

When the report is directed to an extended reporting printer that does not support a Forms Control Block (FCB), the default height of the assigned extended reporting printer defines the height of each line.

`[SPACE` ***`number-of-spaces`*****`]`** Specify SPACE to adjust the default number of blanks (space characters) inserted between fields on TITLE and LINE statement items. The value of *number-of-spaces* has a valid range of 0 to 'nnn' (default is 3), where 'nnn' does not cause line overflow.

When the report is directed to an extended reporting printer, CA-Easytrieve multiplies *number-of-spaces* by the default width of the assigned extended reporting printer. This operation expresses *number-of-lines* in terms of the printer's W-unit.

Note: The SPREAD parameter overrides this parameter.

[TITLESKIP *number-of-lines*] Specify TITLESKIP to insert blank lines between the last title line and the first heading line (or LINE 01) of a report. The value of *number-of-lines* has a valid range of 0 to 'nnn,' where 'nnn' allows for the printing of at least one line group on each page.

When the report is directed to an extended reporting printer that does not support a Forms Control Block (FCB), the height of each line is defined by the default height of the assigned extended reporting printer. This operation converts *number-of-lines* into the H-units applicable to the printer.

[CONTROLSKIP *number-of-lines*] (Mainframe and UNIX only) Specify CONTROLSKIP to define the number of blank lines to be inserted following CONTROL total lines and the next detail line. *Number-of-lines* must be between 0 and 32767. If CONTROLSKIP is not specified, one blank line plus the SKIP value is inserted after the CONTROL total line.

```
[SPREAD  ]
[NOSPREAD]
```

Specify SPREAD to insert the maximum number of spaces between each column of a report. NOSPREAD deactivates SPREAD when it is the default specified in the Site Options Table. SPREAD and NOADJUST are mutually exclusive. See Report Processing in the *CA-Easytrieve Programmer Guide* for more information and examples of this parameter.

Note: SPREAD overrides the SPACE parameter.

[NOADJUST] Specify NOADJUST to left-justify the title lines and report on the page. The default is centered on the page. SPREAD and NOADJUST are mutually exclusive.

```
[NODATE   ]
[LONGDATE ]
[SHORTDATE]
```

Specify NODATE to inhibit the printing of the date value on the first title line (TITLE 01).

LONGDATE specifies that SYSDATE-LONG appears on the first title line.

SHORTDATE specifies that SYSDATE appears on the first title line.

[NOPAGE] Specify NOPAGE to inhibit the printing of the value of PAGEWRD (in the Site Options Table) and the current page number in the report title.

[NOHEADING] Specify NOHEADING to inhibit the printing of column headings. The default is that each field's HEADING value is printed as a column heading.

Testing Aid Parameters

LIMIT and EVERY are used as testing aids for report development. These parameters control the amount of data output on a report.

`[LIMIT` ***`number-of-records`*****`]`** Specify LIMIT to limit the number of records processed by the report. The value of *number-of-records* has a valid range of 1 to 32,767.

`[EVERY` ***`n-number-of-lines`*****`]`** Specify EVERY to indicate that only every n line is printed in the report. The value of *n-number-of-lines* has a valid range of 1 to 32,767.

Usage Notes

The data window for fields with VARYING specified on the DEFINE statement is based on the maximum length of the field. The window is padded to the right with blanks for VARYING fields less than the maximum.

You need not code the SUMMARY parameter to use SUMFILE.

See the *CA-Easytrieve Programmer Guide* for a complete explanation of CA-Easytrieve reporting facilities.

REPORT-INPUT Report Procedure

A REPORT-INPUT procedure selects or modifies report input data.

Syntax

```
REPORT-INPUT. PROC
```

Usage Notes

This procedure is performed for each PRINT statement (report input). To cause the data to continue into report processing, you must execute a SELECT statement for the associated input data. In other words, input that does not get SELECTed is bypassed for continued processing.

Although you can code the logic to select records in the JOB activity itself, you can occasionally place the logic in a REPORT-INPUT procedure.

When the report data has been spooled (because the report had been SEQUENCEd or the printer file was in use), the REPORT-INPUT procedure is invoked as each spooled record is read to produce this report. This means that all records PRINTed are spooled and sorted (if SEQUENCE is specified). The REPORT-INPUT procedure is then invoked. For performance reasons, you should select the records in a JOB activity, if possible.

A REPORT-INPUT procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

The following example illustrates use of the REPORT-INPUT procedure in final report input selection. Only the first record in each ZIP code is selected.

Statements:

```
FILE FILE1
LAST-NAME  1  5 A
STATE      6  2 A
ZIP        8  5 N
PAY-NET    13 5 N 2
HOLD-ZIP    S 5 N VALUE 00000
JOB INPUT FILE1 NAME MYPROG
  PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65 +
  SUMMARY  SUMCTL DTLCOPY
  SEQUENCE STATE ZIP
  CONTROL  STATE NEWPAGE ZIP
  TITLE 'REPORT FOR THE STATE OF' STATE
  LINE 01  LAST-NAME STATE ZIP PAY-NET
*
REPORT-INPUT. PROC
  IF ZIP NE HOLD-ZIP
    HOLD-ZIP = ZIP
    SELECT
  END-IF
END-PROC
*
```

Data:

```
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
```

Results:

```
 11/23/86          REPORT FOR THE STATE OF   IL      PAGE      1

             LAST-NAME   STATE    ZIP     PAY-NET
               BROWN      IL     60076      123.45
               JONES      IL     60077       98.76
                          IL                222.21

 11/23/86          REPORT FOR THE STATE OF   TX      PAGE      2

             LAST-NAME   STATE    ZIP     PAY-NET
               SMITH      TX     75218      111.11
                          TX                111.11

                                            333.43
```

RESHOW Statement

The RESHOW statement is used in an AFTER-SCREEN procedure to re-display the screen image with user-entered data intact. In contrast to the REFRESH statement, the screen image is not rebuilt using the current values of program fields.

Syntax

```
RESHOW
```

Usage Notes

Upon receiving the screen, CA-Easytrieve saves a copy of the screen image. The RESHOW statement restores the saved image.

Example

As shown in the following example, RESHOW can be used to re-display a screen following a request for help. The data that the user entered on the screen before he requested help is re-displayed intact. When RESHOW is used with an IMMEDIATE KEY, original screen data is retained, but not edited or saved into program fields.

```
...
SCREEN NAME MENU UPPERCASE
  KEY ENTER
  KEY F1 NAME 'Help' IMMEDIATE
  KEY F3 NAME 'Exit' EXIT
  TITLE...
  ROW...
  AFTER-SCREEN. PROC
    IF KEY-PRESSED = F1
      EXECUTE MENU-HELP
      RESHOW
    END-IF
    CASE OPTION
    ...
  END-PROC
SCREEN NAME MENU-HELP
  KEY F3 NAME 'Exit' EXIT
  TITLE...
  ROW...
```

RETRIEVE Statement (CA-IDMS and IMS/DLI)

The RETRIEVE statement identifies the database records that are automatically input to the JOB activity. Code the RETRIEVE statement immediately following a JOB statement to specify automatic input. You can code only one RETRIEVE statement in each JOB activity. CA-Easytrieve processes the automatic input the same way as non-database input.

Syntax

Format 1 (CA-IDMS)

```
RETRIEVE  file-name  +

 [             {program-name     }]
 [PROGRAM-NAME {                 }]  +
 [             {'program-literal'}]

 [       {db-name-table-name     }]
 [DBNAME {                       }]  +
 [       {'db-name-table-literal'}]

 [     {node-name     }]
 [NODE {              }]  +
 [     {'node-literal'}]

 [         {dictionary-name     }]
 [DICTNAME {                    }]  +
 [         {'dictionary-literal'}]

 [         {dictionary-node-name     }]
 [DICTNODE {                         }]  +
 [         {'dictionary-node-literal'}]

 [KEYFILE tickler-file-name +                                 ]   ]
 [KEYVALUE (calc-key-field-name EQ calc-value-field-name ...) ] + ]
 [                                                                ] +
 [DUPS                                                            ]
 [NODUPS                                                          ]

 SELECT (record-name  +

        [AREA  'area-literal'  +                                    ]
        [SET   'set-literal'  +                                     ] +
        [INDEX 'index-set-literal' [USING ('index-key-literal' ...)]]

        [ID 'path-literal']  +

        [LIMIT number-of-records] +

        [WHILE (condition)]  +
                  ...)
```

Format 2 (IMS/DLI)

```
RETRIEVE  file-name  +

  [KEYFILE tickler-file-name  KEYVALUE key-field-name]  +

  SELECT (record-name  +

         [ID  'path-literal']  +

         [LIMIT number-of-records]  +

         [SSA 'segment-literal']  +

         [WHILE (condition)]  +

         ...)
```

Parameters

Format 1 (CA-IDMS)

`file-name` *File-name* is the same as the name coded in the FILE *file-name* IDMS and JOB INPUT *(file-name)* statements.

```
[             {program-name     } ]
[PROGRAM-NAME {                 } ]
[             {'program-literal'} ]
```

Program-name or *'program-literal'* specifies the name used to identify the program to CA-IDMS during execution. *Program-name* must be an eight-byte alphanumeric field. *'Program-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[       {db-name-table-name     } ]
[DBNAME {                       } ]
[       {'db-name-table-literal'} ]
```

Db-name-table-name or *'db-name-table-literal'* specifies a DB Name Table. Data retrieved during execution of the user's program is from the named CA-IDMS database. *Db-name-table-name* must be an eight-byte alphanumeric field. *'Db-name-table-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[     {node-name     } ]
[NODE {              } ]
[     {'node-literal'} ]
```

Node-name or *'node-literal'* specifies the Central Version Node that hosts the CA-IDMS activity generated by the user's program. *Node-name* must be an eight-byte alphanumeric field. *'Node-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[         {dictionary-name     } ]
[DICTNAME {                    } ]
[         {'dictionary-literal'} ]
```

Dictionary-name or *'dictionary-literal'* specifies the Dictionary Name of a Secondary Load Area. *Dictionary-name* must be an eight-byte alphanumeric field. *'Dictionary-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[         {dictionary-node-name      } ]
[DICTNODE {                          } ]
[         {'dictionary-node-literal' } ]
```

Dictionary-node-name or *'dictionary-node-literal'* specifies the Dictionary Node of a Secondary Load Area. *Dictionary-node-name* must be an eight-byte alphanumeric field. *'Dictionary-node-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[KEYFILE tickler-file-name +                                  ]   ]
[ KEYVALUE (calc-key-field-name EQ calc-value-field-name ...) ] + ]
[                                                                 ]
[DUPS  ]                                                          ]
[NODUPS]                                                          ]
```

The optional tickler file is designated by coding the KEYFILE and KEYVALUE parameters. *Tickler-file-name* is the name of a file that is sequentially processed to obtain the keys of the root records to be retrieved. The DBCS code system of *tickler-file-name* must equal the DBCS code system of *file-name*.

Calc-value-field-name is a data field from *tickler-file-name* that contains a value for one of the CALC keys of the root record. *Calc-key-field-name* is a CALC key field defined in the RECORD statement for the root record that is to receive the value of *calc-value-field-name*. *Calc-value-field-name* is assigned to *calc-key-field-name* using the rules in the "Coding a CA-Easytrieve Program" chapter in the *CA-Easytrieve Programmer Guide*.

You must code one *calc-value-field-name* for each key field defined in the KEY parameter of the RECORD statement for the root record.

The key values are used in the CALC retrieval of root records. Therefore, only CALC records can be root records when the tickler file is used. The optional keywords, DUPS and NODUPS, are used to specify whether CALC records with duplicate keys are also retrieved. The OPTIONS table parameter CALCDUP has the default value. The JOB activity is terminated at end-of-file for *tickler-file-name*.

The KEY parameter for the root record retrieved by the tickler file option must be specified on the RECORD statement.

[SELECT (*record-name* ...)] The SELECT parameter identifies which paths are retrieved. *Record-name* must be the same as coded on a RECORD statement. Any number of records and paths can be coded under control of the following rules of network structure:

- The first *record-name* coded is the root. It is retrieved by an area sweep, tickler file, or integrated index.

- A repeated *record-name* denotes a node in the network. A node is a record-type that is common in multiple paths. The optional subparameters are not allowed when a *record-name* is repeated as a node.
- Paths are retrieved in the order in which they are identified.

`[AREA 'area-literal']` The optional AREA subparameter is coded to supply the sweep area. This subparameter can be specified only if *record-name* is a root record. AREA is not allowed if INDEX has already been specified for this record. The one to sixteen-character CA-IDMS area name (*'area-literal'*) controls retrieval within area of root records. *'Area-literal'* must be alphanumeric (non-DBCS), and is padded to the right (if necessary) to create a 16-byte value.

If the AREA subparameter is coded, CA-Easytrieve uses OBTAIN NEXT *record-name* WITHIN AREA calls to retrieve occurrences of this record. If this subparameter is omitted, CA-Easytrieve uses OBTAIN NEXT *record-name* calls instead.

`[SET 'set-literal']` The SET subparameter specifies the name of the set used for retrieving the named record (*record-name*). This subparameter is not allowed if *record-name* is the root record or if *record-name* is a node. SET is required for all other records. *'Set-literal'* must be alphanumeric (non-DBCS), and is padded to the right (if necessary) to create a 16-byte value.

If this record is a member of the specified set, CA-Easytrieve uses OBTAIN NEXT *record-name* WITHIN SET calls to retrieve occurrence of this record. If this record is the owner of the specified set, CA-Easytrieve uses OBTAIN OWNER calls instead.

`[INDEX 'index-set-literal'[USING ('index-key-literal' ...)]]` Code the optional INDEX subparameter to designate the index set (*'index-set-literal'*) that controls root retrieval by integrated indexing. This subparameter can be specified only if *record-name* is a root record. INDEX is not allowed if AREA has already been specified for this record. *'Index-key-literal'* must be a alphanumeric (non-DBCS), and is padded to the right (if necessary) to create a 16-byte value.

Note: The INDEX subparameter cannot be used with the tickler file.

The optional USING subparameter (*'index-key-literal'*) designates the alphanumeric literals used to constrain the index. You can code as many occurrences of *'index-key-literal'* as are required to fully specify the index key value. The values are concatenated in the order specified and form the index key value that is passed to CA-IDMS. The cumulative length of all literals specified must match the length of the index known to integrated indexing. The code system of the data must also match.

When the INDEX subparameter is coded, CA-Easytrieve uses OBTAIN NEXT WITHIN SET calls to retrieve all occurrences of the root record except for the first occurrence. The retrieval of the first occurrence is determined by the optional USING subparameter. If the USING subparameter is coded, CA-Easytrieve retrieves the first root record occurrence using an OBTAIN WITHIN SET USING SORT KEY call. If the USING subparameter is omitted, an OBTAIN FIRST WITHIN SET call is used. CA-Easytrieve uses the USING subparameter to establish the initial position within the index set. Once this initial position has been established, retrieval of the root record proceeds until the end of the index set is reached.

`[ID '`*`path-literal`*`']` Code the optional ID subparameter to establish the identity of retrieved paths. The system-defined field *file-name:*PATH-ID is set to the value of *'path-literal'* for the lowest record retrieved in the current path. *'Path-literal'* can be an alphanumeric value of one or two characters. It cannot contain any DBCS data. The default is spaces. Whenever a key of the tickler file does not correspond to a root record in the database, *file-name:*PATH-ID is set to **NF** (Not Found).

`[LIMIT '`*`number-of-records`*`']` The optional LIMIT subparameter controls the number of record occurrences to be retrieved. The limit applies to the specific record in the path. *'Number-of-records'* must be a positive integer. When this subparameter is not coded, all occurrences of the record are retrieved.

`[WHILE (`*`condition`*`)]` Code the optional WHILE subparameter to pre-screen input records. The syntax of the condition is exactly the same as the conditional expressions described in the "Coding a CA-Easytrieve Program" chapter in the *CA-Easytrieve Programmer Guide*. When the associated record is retrieved from CA-IDMS, the condition is evaluated. Records are accepted for input only if the condition is true.

Format 2 (IMS/DLI)

`file-name` *File-name* identifies the database being accessed. *File-name* is the same as the name coded in JOB INPUT *file-name* and FILE *file-name* statements.

`[KEYFILE `*`tickler-file-name`*`  KEYVALUE `*`key-field-name`*`]` You can designate the tickler file option by coding both the KEYFILE and the KEYVALUE parameters. *Tickler-file-name* is the name of the file that is sequentially processed to get the keys of the root segments to be retrieved. *Key-field-name* is a data field from *tickler-file-name* that contains the keys. The key values are used in the segment search argument for the root segment. CA-Easytrieve issues GU (get unique) calls at the root level for each key found in *tickler-file-name*. Automatic input is terminated at end-of-file for *tickler-file-name*.

The DBCS code system assigned to *tickler-file-name* must match the DBCS code system of *file-name*.

`[SELECT (`*`record-name`*` ...)]` The SELECT parameter identifies which segments (*record-name*) CA-Easytrieve retrieves. *Record-name* must be the same as the *segment-name* coded on a RECORD statement. You can identify any number of *record-name*s for input; however, the parent of all selected segments must also be selected.

`[ID '`*`path-literal`*`']` Code the optional ID subparameter to establish the identity of retrieved paths. The system-defined field PATH-ID is set to the value of *'path-literal'* for the lowest segment retrieved in the current path. PATH-ID is a two-byte alphabetic field. *'Path-literal'* can be an alphabetic value of one or two bytes. It cannot contain any DBCS data. The default value for PATH-ID is spaces. When a key of the tickler file does not correspond to a root record in the database, PATH-ID is set to **NF** (Not Found).

`[LIMIT '`*`number-of-records`*`']` The optional LIMIT subparameter controls the number of segment occurrences to be retrieved. The limit applies to each path. *'Number-of-records'* must be a positive integer. When this subparameter is not coded, all occurrences of the segment are retrieved.

`[SSA '`*`segment-literal`*`']` You can code the optional Segment Search Argument (SSA) parameter for the root segment. *'Segment-literal'* is used in the creation of the SSA to qualify segment retrieval. This parameter is not valid when you use a tickler file. The value supplied with SSA is enclosed within parentheses and concatenated with the segment-name to produce the root segment's SSA. *'Segment-literal'* cannot contain any DBCS data.

`[WHILE (`*`condition`*`)]` Code the optional WHILE subparameter to pre-screen input segments. The syntax of the condition is exactly the same as the conditional expressions described in the "Coding a CA-Easytrieve Program" chapter in the *CA-Easytrieve Programmer Guide*. When the associated record is returned by IMS/DLI, the condition is evaluated. Segments are accepted for input only if the condition is true.

Usage Notes

See the CA-Easytrieve Programmer Guide for RETRIEVE statement examples.

ROLLBACK Statement

The ROLLBACK statement causes all uncommitted updates in the current logical unit of work to be rolled back.

Syntax

```
ROLLBACK
```

Usage Notes

See the *CA-Easytrieve Programmer Guide* for more information about types of work that are recoverable. Use the COMMIT statement to commit any pending changes.

Example

```
WRITE PERSNL ADD
...
IF ...
  ROLLBACK
ELSE
  COMMIT
END-IF
```

ROW Statement

The ROW statement specifies the items (fields or literals) to be displayed or received on a row of a screen. Multiple items can be coded on each ROW statement. Attributes can be specified for each literal coded on the ROW statement. Attributes and editing criteria can be specified for each *field-name* coded on the ROW statement.

Syntax

```
ROW [row-number] +

[+offset-value    ] {field-name   }
[                 ] {             } +
[COL column-number] {'row-literal'}

[     {attribute-name  } ]
[ATTR {                } ] +
[     {(attribute-list)} ]

[        {RIGHT} ]
[JUSTIFY {     } ] +
[        {LEFT } ]

[FILL {'fill-character'|NULL}] +

[MASK ({[mask-identifier] [BWZ] ['mask-literal']|HEX})] +
[NOMASK                                               ]

[        {pattern-name} ]
[PATTERN {            } ] +
[        {'pattern'   } ]
```

```
[UPPERCASE] +

[VALUE (literal [THRU literal] [...])] +

[        [     {attribute-name  }]      ] ]
[ERROR ( [ATTR {                }] +    ] ]
[        [     {(attribute-list)}]      ] ]
[                                       ] ] ...
[        [  {'literal'  [   ] }  ]      ] ]
[        [  {           [...] }  ] )    ] ]
[        [  {field-name [   ] }  ]      ] ]
```

Parameters

[*row-number*] *Row-number* specifies the line on which the item on the screen is displayed. A ROW without a *row-number* is assigned the next row number on the screen. Next is defined as the previous *row-number* plus one, not the highest number used as yet.

A ROW without any fields or literals displays a blank line on the screen at the corresponding row-number.

Row-number cannot exceed the default ROWCOUNT value set in the site options or the value of the ROWCOUNT parameter specified on the SCREEN statement, if coded.

```
[+offset-value    ] {field-name   }
[                 ] {             }
[COL column-number] {'row-literal'}
```

The +*offset-value* or the COL *column-number* parameter allows you to control positioning of an item on the row.

+*Offset-value* is the number of columns (spaces) preceding a screen item. The default +*offset-value* is +1 because the space preceding each screen item is reserved for screen attributes. +*Offset-value* must be a signed positive integer and can only be used for items other than the first in the row.

Use *column-number* to explicitly specify the column at which the screen item is displayed.

If you do not code an +*offset-value* or *column-number*, the next *field-name* or *'row-literal'* is displayed one column after the end of the previous *field-name* or *'row-literal.'* If no previous item exists in the row, the item is displayed in column one.

Field-name can be any DEFINEd field in your program.

'Row-literal' can be any text you want to display on the screen.

The sum of the length of all screen items (fields and literals) plus *offset-values* and *column-numbers* (if used) cannot exceed the value of the default LINESIZE set in the site options, or the value of the LINESIZE parameter on the SCREEN statement, if coded.

```
[     {attribute-name  } ]
[ATTR {                } ]
[     {(attribute-list)} ]
```

ATTR specifies either a DECLAREd screen attribute name or one or more attribute keywords. See the ATTR Parameter for a list of attributes. See the DECLARE Statement for more information on DECLAREd screen attributes.

The following attributes are invalid for literals and system-defined read-only fields:

- CURSOR
- NUMERIC
- INVISIBLE
- MUSTFILL
- MUSTENTER
- TRIGGER
- ALARM

They are ignored if used, but CA-Easytrieve issues a warning message during compilation.

SENDONLY and ASKIP are assumed for literals and system-defined read-only fields.

```
[        {RIGHT} ]
[JUSTIFY {     } ]
[        {LEFT } ]
```

Use the JUSTIFY parameter to specify whether the data in the field is left or right justified when displayed at the terminal.

[FILL {'*fill-character*'|NULL}] Specify FILL to translate trailing blanks into either *'fill-character'* or NULL. *'Fill-character'* must be a one-byte alphanumeric literal.

Upon receiving data from the screen, CA-Easytrieve translates all remaining fill characters to spaces.

You can use the FILL parameter to fill a field with underscores to illustrate the total length of the field. You can fill a field with NULL on a 3270 display to allow insertion of characters.

Varying length fields with FILL NULL do not have trailing nulls translated to spaces. The first trailing null terminates the varying length field, and then sets its length.

```
[MASK ({[mask-identifier] [BWZ] ['mask-literal']|HEX})]
[NOMASK                                              ]
```

The optional MASK parameter is used to format the field for display.

If MASK is not coded, the MASK coded on the field's definition is used. Use NOMASK to specify that the field's default MASK is to be used instead of the field's definition MASK.

Any letter from A through Y can be used as an optional *mask-identifier*. You can use the letter to identify a new mask or to retrieve a mask that was previously defined either in the Site Options Table or by a mask parameter on a previous field definition or ROW usage. If the new mask that you identify does not already exist, CA-Easytrieve retains the mask for future reference. Do not use the same identifier to establish more than one mask.

The BWZ (blank when zero) option suppresses the display of *field-name* when it contains all zeros. BWZ can be used by itself or with other options on the MASK parameter.

'Mask-literal' defines an edit mask and must be enclosed within single quotes. The actual edit mask is coded according to the rules specified under the MASK Parameter.

Specify HEX to display the field in double-digit hexadecimal format. You can display fields of up to 50 bytes with the HEX mask.

When fields are received from the terminal, the mask is used as an editing template also. Special characters in the MASK are stripped from the data before it is moved into the field data area. See the *CA-Easytrieve Programmer Guide* for more information.

Note: HEX edit masks are not allowed for VARYING fields.

```
[        {pattern-name} ]
[PATTERN {              } ]
[        {'pattern'     } ]
```

PATTERN allows you to specify a pattern against which each input character is edited. The pattern can be specified as a literal or as the name of a DECLAREd pattern. See the DECLARE Statement.

The valid pattern characters and their meanings are shown in the following table.

Character	Meaning
A	Represents a lower-case or an upper-case letter

Character	Meaning
B	Represents a single blank
D	Represents a digit
E	Represents an empty string
L	Represents a lower-case letter
N	Represents an upper-case letter or a national character
U	Represents an upper-case letter
X	Represents any character
"x"	Double quotes surrounding a character or a sequence of characters literally represent the character or sequence of characters contained within. The x represents any character. To literally represent single or double quotes, use two sets of quotes within the surrounding set of double quotes (‘""""‘ or ‘"x""x"‘, ‘"‘‘"‘, or ‘"x’‘x"‘).
blank	Blanks (unless contained in double quotes) serve as delimiters but are otherwise ignored. They can be inserted into the pattern to increase readability.
( )	Represents grouping to control the precedence of operators.
or | or ,	Represents a choice (or alternation operator).
(m) or (m..n) or (m..*) or (*) or *	Represents the repetition of the preceding pattern expression. The m and n represent numbers and m must be less than n. A single number with parentheses indicates the exact number of repetitions. (m..n) represents a range of repetitions, minimum to maximum. An asterisk in a range, (m..*), represents an infinite maximum. An asterisk by itself, (*) or *, represents a range from 0 to infinity.
# or /-/	Represents the remove (or toss) operation. This operation applies only to a single character set at a time and must immediately follow that character set in the pattern. This operation removes the character that matched the character set from the data.
+	Represents character set addition to form another character set.
-	Represents character set difference to form another character set.
concatenation	Concatenation is implied by proximity. For example, DDDU means 3 digits followed by an upper-case letter.

The precedence of operators from highest to lowest:

```
Grouping:                        () " "
Set construction:                + -
Actions:                 #
Repetition:              (n) (m..n) (m..*) (*)
Concatenation:           proximity
Choice:                  |
```

The edit pattern is evaluated from left to right (the data from the screen is processed from left to right). Patterns examine only one character at a time. They do not look ahead and they do not back track. See the *CA-Easytrieve Programmer Guide* for more information.

[UPPERCASE] Specify UPPERCASE to translate the field coming from the terminal to upper case before placing it in the field data area.

[VALUE (*literal* [THRU *literal*] [...])] Use VALUE to specify a value, a series of values, or a range of values (or a combination) that constrain the values accepted in the field. Values must be specified as literals of the correct type for the field. See Conditional Expressions in the “Statements A–C” chapter for more information.

```
[        [     {attribute-name  }    ]
[ERROR ( [ATTR {                } +  ]
[        [     {(attribute-list)}    ]
[                                    ]
[        [  {'literal'  [   ] }  ]   ]
[        [  {           [...] }  ] ) ]
[        [  {field-name [   ] }  ]   ]
```

ERROR specifies one or more fields or alphanumeric literals to be used as the error message issued by CA-Easytrieve in case of an automatically-detected error condition. The total length of the message text cannot exceed the current screen size less two or the compiler issues an error message. Optionally, you can specify the screen attribute to be used for the field in error.

Usage Notes

See the *CA-Easytrieve Programmer Guide* for more information.

Example

```
ROW 5 'Number'  COL 20 EMP-NUMBER  ATTR PROTECT  MASK 'ZZ9'
ROW   'Name'    COL 20 EMP-NAME    UPPERCASE
ROW   'Dept'    COL 20 EMP-DEPT    VALUE (900 THRU 999) +
                                   ERROR 'Invalid Department'
```

Chapter

7

Statements S - Z

SCREEN Statement

The SCREEN statement defines and initiates a SCREEN activity. A SCREEN activity defines a transaction-oriented processing activity under the control of keys pressed by the terminal operator. Statements can also be inserted in screen procedures to retrieve and maintain files and databases.

Note: Screen processing is not available in the UNIX environment.

Syntax

```
                          [        [ACTIVITY  ] [TERMINAL  ] ]
SCREEN [NAME screen-name] [COMMIT ([          ] [          ])] +
                          [        [NOACTIVITY] [NOTERMINAL] ]

      [UPPERCASE] [ROWCOUNT rows] [LINESIZE columns] +

      [ROW screen-start-row] [COL screen-start-column] +

      [                {attribute-name  } ]
      [BACKGROUND ATTR {                } ]  +
      [                {(attribute-list)} ]

      [        {SINGLE          }                            ]
      [        {DOUBLE          } [     {attribute-name  } ] ]
      [BORDER ({                } [ATTR {                } ] ] +
      [        {WIDE            } [     {(attribute-list)} ] ]
      [        {'border literal'}                            ]

      [SHADOW]
```

Parameters

`[NAME screen-name]` Optionally, specify a name for the SCREEN activity. *Screen-name* can:

- Be up to 128 characters in length
- Contain any character other than a delimiter

- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

The *screen-name* can be used to identify the screen in an EXECUTE statement.

```
[        [ACTIVITY  ]  [TERMINAL  ] ]
[COMMIT ([          ]  [          ])]
[        [NOACTIVITY]  [NOTERMINAL] ]
```

Specify the COMMIT parameter to control the logical unit of work. COMMIT indicates when the activity commits recoverable work. Each commit point posts all updates, additions and deletions, terminates holds, and closes SQL cursors.

Specify ACTIVITY to commit all recoverable work during the normal termination of the activity. Specify NOACTIVITY to tell CA-Easytrieve not to commit at the end of the activity. NOACTIVITY is the default.

Specify TERMINAL to commit all recoverable work during any terminal I/O operation. In CICS, this results in terminal I/O being performed in a pseudo-conversational mode. Specify NOTERMINAL to tell CA-Easytrieve not to commit during a terminal I/O. TERMINAL is the default.

If this activity is executed by an activity that has NOTERMINAL specified, this activity performs terminal I/O as if NOTERMINAL was specified.

Note: You can also issue your own COMMIT and ROLLBACK statements to commit or recover on a controlled basis.

`[UPPERCASE]` Specify UPPERCASE to translate the data received from the terminal to upper case before it is processed. If UPPERCASE is not specified, the data is processed as the user enters it.

`[ROWCOUNT` ***`rows`*****`]`** ROWCOUNT *rows* allows you to override the default number of terminal rows for the screen display. The default is set in the Site Options Table. See LINESIZE columns, below, for valid ROWCOUNT-LINESIZE combinations.

`[LINESIZE` ***`columns`*****`]`** LINESIZE *columns* allows you to override the default number of columns for the screen display. The default is set in the Site Options Table.

On the **mainframe**, ROWCOUNT can be any value from 1 to 255. LINESIZE can be any value from 1 to 255. If the dimensions of the screen exceed the screen size available on the display terminal, only a portion of the screen is displayed.

On the **workstation**, ROWCOUNT can be any value from 1 to 25. LINESIZE can be any value from 1 to 80. A ROWCOUNT of 25 is flagged as an error if the portability compiler switch (/P) is specified. See the “Compiling From the Command Line” chapter in the *CA-Easytrieve/Workstation User Guide* for more information.

`[ROW` *`screen-start-row`*`] [COL` *`screen-start-column`*`]` *Screen-start-row* specifies the starting row of the screen. The default is **1**.

Screen-start-column specifies the starting column of the screen. The default is **1**.

```
[                {attribute-name  } ]
[BACKGROUND ATTR {                } ]
[                {(attribute-list)} ]
```

(Workstation only) Use BACKGROUND to specify a DECLAREd screen attribute name or a list of attribute keywords for the background of the workstation screen display. All attributes except colors and INTENSE are ignored. If not specified, the default background color is BLACK. See the ATTR Parameter for a list of attribute keywords. See the DECLARE Statement for more information on DECLAREd screen attributes.

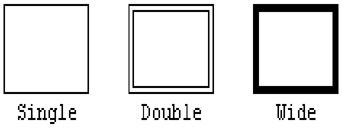
BACKGROUND is ignored on the mainframe.

```
[        {SINGLE          }                             ]
[        {DOUBLE          } [     {attribute-name  } ] ]
[BORDER ({                } [ATTR {                } ] ]
[        {WIDE            } [     {(attribute-list)} ] ]
[        {'border literal'}                             ]
```

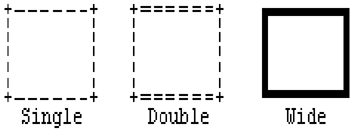
Use BORDER to specify that the screen has a border.

SINGLE, DOUBLE, or WIDE specify that the border is built from a pre-defined line-drawing character set.

Borders on the workstation are:



Borders on the mainframe are:



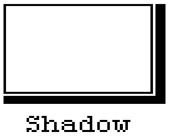
'Border literal' specifies the character to be used for the screen border. This value must be a single character enclosed in single quotes.

Optionally, specify a DECLAREd screen attribute name or a list of attribute keywords for the screen border. The following attributes are ignored for BORDER:

- CURSOR
- NUMERIC
- INVISIBLE
- MUSTFILL
- MUSTENTER
- TRIGGER
- ALARM

See the ATTR Parameter for a list of attribute keywords. See the DECLARE Statement for more information on DECLAREd screen attributes.

`[SHADOW]` (Workstation only) Specify SHADOW to display a shadow on the bottom and right of the screen, as shown:



SHADOW is ignored on the mainframe.

Usage Notes

The structure of a SCREEN activity is as follows:

```
SCREEN statement
  Screen declaration statements:
    DEFAULTs (first in declaration section)
    KEYs, TITLEs, ROWs (in any order)
  Screen procedures (both special-named and user-defined, in                            any
order)
```

SCREEN activities can be EXECUTEd by PROGRAM or other SCREEN activities. If a PROGRAM activity is not present, the first SCREEN activity detected is automatically executed. A SCREEN activity continues processing until an EXIT, STOP, or TRANSFER statement is executed. CA-Easytrieve issues an error message when compiling a screen activity that does not contain one of these statements.

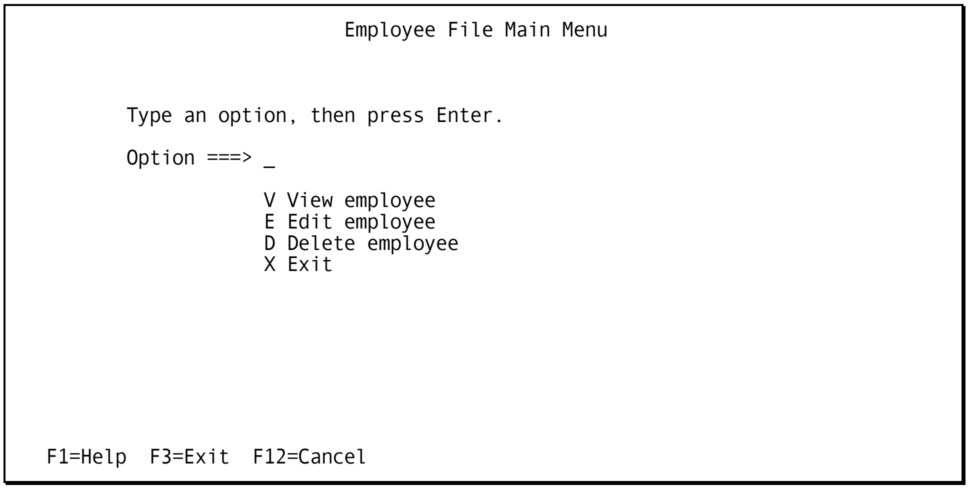
If the LINESIZE and ROWCOUNT for a screen are less than the line size and number of rows on the terminal, the screen is displayed as a pop-up window. Any fields from previous screens that are still displayed are given the ASKIP attribute to prevent data entry on those screens.

When executing in TSO and CMS, if the terminal supports two presentation sizes, CA-Easytrieve selects the presentation size based on the size of the screen. When a pop-up window is displayed, the presentation space is based on the larger of the previous display size or the size of the pop-up window.

Example

```
DEFINE WS-REPLY  W 1 A
SCREEN NAME MAIN-MENU
  TITLE 'Employee File Main Menu'
  ROW  6 COL 10 'Type an option, then press Enter.'
  ROW  8 COL 10 'Option ===>' WS-REPLY VALUE ('V' 'E' 'D' 'X') +
                  ERROR 'Please type V, E, D, or X'
  ROW 10 COL 22 'V View employee'
  ROW    COL 22 'E Edit employee'
  ROW    COL 22 'D Delete employee'
  ROW    COL 22 'X Exit'
  KEY F1  NAME 'Help' IMMEDIATE
  KEY F3  NAME 'Exit' EXIT
  KEY F12 NAME 'Cancel' EXIT IMMEDIATE
```

The following illustrates the screen created from the example screen declaration:



```
                             Employee File Main Menu



         Type an option, then press Enter.

         Option ===> _

                     V View employee
                     E Edit employee
                     D Delete employee
                     X Exit






   F1=Help  F3=Exit  F12=Cancel
```

SEARCH Statement

The SEARCH statement provides access to table information. Special conditions of the IF statement can be used to validate the results of SEARCH operations.

Syntax

```
SEARCH file-name  WITH search-field  GIVING result-field
```

Parameters

`file-name` *File-name* is the name of the file that describes the table and its source. The file must have the TABLE parameter on its FILE statement and must be a fixed length.

`WITH` ***`search-field`*** *Search-field* identifies the field containing the search argument for the binary search. This parameter is defined in any file, except for files with the TABLE parameter or it can be defined in working storage.

The length and field type of *search-field* must match the length and field type of the ARG field defined for *file-name*. *Search-field* cannot be a varying length field or a nullable field.

`GIVING` ***`result-field`*** *Result-field* identifies the receiving field for the results of the table search. This parameter is defined in any file, except for files with the TABLE parameter or it can be defined in working storage.

The length and field type of *result-field* must match the length and field type of the DESC field defined for *file-name*. *Result-field* cannot be a varying length field or a nullable field.

Usage Notes

After each SEARCH statement, you can code an IF *file-name* test to determine the success of the table search. When the search is successful (IF *file-name* is true), *result-field* contains the table's descriptive data corresponding to the search argument of *search-field*. When the search is unsuccessful (IF *file-name* is false), the contents of *result-field* are unchanged.

You can code SEARCH statements any place in a PROGRAM, SCREEN, or JOB activity, and issue any number of SEARCHes against any number of tables.

The file must be in ARG sequence and cannot contain any duplicates. The compare between the WITH field and the ARG field in the table is a logical compare, that is, the compare ignores the data type and treats both fields as if they have a data type of A.

When the table file is also an INDEXED file and the ARG field is the key, CA-Easytrieve performs a keyed read of the file. Otherwise, the entire file is read into memory and a binary search is performed. See the *CA-Easytrieve Programmer Guide* for more information on table processing.

Example

The following example illustrates the retrieval of high school class descriptions based upon class identification codes.

Statements:

```
DEFINE CODE          W   4 A
DEFINE DESCRIPTION   W  40 A
FILE CLASSES TABLE INSTREAM
ARG   1  4 A
DESC 10 40 A
1011     ENGLISH I
1012     ENGLISH II
1013     ENGLISH III
1014     ENGLISH IV
ENDTABLE
PROGRAM NAME MYPROG
  MOVE '1012' TO CODE
  SEARCH CLASSES WITH CODE, GIVING DESCRIPTION
  IF CLASSES
    DISPLAY DESCRIPTION
  ELSE
    DISPLAY 'CLASS NOT FOUND'
  END-IF
```

Results:

```
 ENGLISH II
```

SELECT Statement (File-based SQL)

A SELECT statement issued for an SQL file causes a cursor to be automatically declared and opened as a CA-Easytrieve file. The resulting cursor can then be fetched and updated by subsequent commands for the file. The cursor can also be the subject of automatic input using the JOB statement.

Syntax

```
SELECT [DISTINCT] [FROM] file-name +

 [WHERE search-condition] +

 [GROUP BY column-name        + ] +
 [         [, column-name ...]  ]

 [HAVING search-condition] +

 [         {column-name} [ASC ]         ]
 [ORDER BY {           } [    ] +       ]
 [         {integer    } [DESC]         ]
 [                                      ]
 [         [  {column-name} [ASC ]    ] ]
 [         [, {           } [    ] ...] ] +
 [         [  {integer    } [DESC]    ] ]
```

```
[FOR UPDATE]
```

Parameters

[DISTINCT] DISTINCT eliminates duplicate rows. If DISTINCT is not specified, all rows are retrieved.

[FROM] ***file-name*** Optionally, code FROM for statement readability.

File-name must be the name of a CA-Easytrieve SQL file.

[WHERE ***search-condition*****]** *Search-condition* is used to specify conditions for the retrieval of data. The *search-condition* is applied to create the result set for the file. Refer to your SQL vendor manuals for a description of the *search-condition*.

[GROUP BY ***column-name*****]** GROUP BY is used to group data which is FETCHed into the file. See your SQL vendor's guide for *column-name* syntax.

[HAVING ***search-condition*****]** *Search-condition* is used to specify the data to be provided to the user. HAVING can be used to compare the results of all the returned data with a specific value in the data provided (such as the minimum or maximum value). Refer to your SQL vendor manuals for a description of the *search-condition*.

```
[         {column-name} [ASC ] ]
[ORDER BY {           } [    ] ]
[         {integer    } [DESC] ]
```

ORDER BY returns the rows of the result table in the order of the values of the specified *column-names*. *Integer* references a column by its position in the result table rather than by a *column-name*. ASC returns the rows in ascending order and is the default. DESC returns the rows in descending order.

[FOR UPDATE] Specify FOR UPDATE to allow updates of the updatable fields defined in *file-name*. If used, FOR UPDATE must be the last parameter specified on the SELECT statement. If FOR UPDATE is not coded and you attempt to update *file-name*, you receive an error at execution.

Usage Notes

- If no *SELECT* statement is issued for an SQL file, a default *SELECT* is used (*SELECT* all defined columns *FROM file-name*).
- If *SELECT* is the first statement in a *JOB* activity, the following happens:
  - If the SELECT is for an automatic input file, the SELECT overrides the default SELECT.

  - If the SELECT is for a file not used for automatic input, a DEFER should be coded on the SQL FILE statement. If DEFER is not coded, the default SELECT is opened during the initialization processing, then closed, and the coded SELECT processed. This causes unnecessary processing to occur.

- If a *SELECT* is specified for a file that has already been opened, either by the default SELECT or another coded SELECT, then the existing *SELECT* for the file is closed and the new *SELECT* is used to open the file again.
- *SELECT* can be coded in a *JOB*'s *START* procedure. However, since a file is normally opened before invoking the *START* procedure, you should specify *DEFER* on the *FILE* statement. Otherwise, the default *SELECT* is opened before the *START* procedure, and then the *SELECT* in the *START* procedure closes the default *SELECT* before it opens. This causes extra processing that is not needed.

Examples

The following is a file-based SQL SELECT statement example:

```
FILE PERSNL SQL (PERSONNEL)
EMPNAME       *   20  A
WORKDEPT      *   2   P    0
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL
  SELECT FROM PERSNL WHERE WORKDEPT = 921
  DISPLAY EMPNAME +2 WORKDEPT
```

The next example shows a file-based SQL SELECT statement with DEFER:

```
FILE PERSNL SQL (PERSONNEL) DEFER
EMPNAME       *   20  A
WORKDEPT      *   2   P    0
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL START START-PROC
  DISPLAY EMPNAME +2 WORKDEPT
START-PROC. PROC
  SELECT FROM PERSNL WHERE WORKDEPT = 921
END-PROC
```

SELECT Statement (CA-IDMS)

(Workstation only) Code a SELECT statement immediately following a JOB statement to specify automatic input of logical records from CA-IDMS databases. You can code only one SELECT statement in each JOB activity. This statement identifies the logical record that is input to the JOB activity. The arguments of the WHERE parameter are passed to CA-IDMS for evaluation.

Syntax

```
  SELECT logical-record-name  +
[WHERE (boolean-expression)]  +

[LIMIT record-literal]  +

[             {program-name     }]
[PROGRAM-NAME {                 }]  +
[             {'program-literal'}]

[       {db-name-table-name     }]
[DBNAME {                       }]  +
[       {'db-name-table-literal'}]

[     {node-name     }]
[NODE {              }]  +
[     {'node-literal'}]

[         {dictionary-name     }]
[DICTNAME {                    }]  +
[         {'dictionary-literal'}]

[         {dictionary-node-name     }]
[DICTNODE {                         }]
[         {'dictionary-node-literal'}]
```

Parameters

`logical-record-name` *Logical-record-name* is a one to sixteen-character name that identifies the logical record to be retrieved. *Logical-record-name* must be the name of a logical record defined by a LOGICAL-RECORD statement.

`[WHERE (boolean-expression)]` Code the optional WHERE clause to provide a Boolean expression that CA-IDMS evaluates before selecting logical records for CA-Easytrieve. Records are sent to CA-Easytrieve only if the Boolean expression is true.

`[LIMIT record-literal]` The optional LIMIT subparameter controls the number of occurrences of the logical record to be retrieved. *Record-literal* must be a positive integer. When this subparameter is not coded, all occurrences of the logical record are retrieved.

```
[             {program-name     }]
[PROGRAM-NAME {                 }]
[             {'program-literal'}]
```

Program-name or *'program-literal'* specifies the name used to identify the program to CA-IDMS during execution. *Program-name* must be an eight-byte alphanumeric field. *'Program-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[       {db-name-table-name     }]
[DBNAME {                       }]
[       {'db-name-table-literal'}]
```

Db-name-table-name or *'db-name-table-literal'* specifies a DB Name Table. Data retrieved during execution of the user's program is from the named database. *Db-name-table-name* must be an eight-byte alphanumeric field. *'Db-name-table-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[     {node-name     }]
[NODE {              }]
[     {'node-literal'}]
```

Node-name or *'node-literal'* specifies the Central Version Node that hosts the CA-IDMS activity generated by the user's program. *Node-name* must be an eight-byte alphanumeric field. *'Node-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[         {dictionary-name     }]
[DICTNAME {                    }]
[         {'dictionary-literal'}]
```

Dictionary-name or *'dictionary-literal'* specifies the Dictionary Name of a Secondary Load Area. *Dictionary-name* must be an eight-byte alphanumeric field. *'Dictionary-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

```
[         {dictionary-node-name     }]
[DICTNODE {                         }]
[         {'dictionary-node-literal'}]
```

Dictionary-node-name or *'dictionary-node-literal'* specifies the Dictionary Node of a Secondary Load Area. *Dictionary-node-name* must be an eight-byte alphanumeric field. *'Dictionary-node-literal'* must be alphanumeric and is padded to the right (if necessary) to create an eight-byte value.

Example

```
*
  IDD SUBSCHEMA DEMOSSLR SCHEMA DEMOSCHM +
      SELECT (CUST-SALES-LR)
*
  JOB INPUT DEMOSSLR
     SELECT CUST-SALES-LR
*
     DISPLAY CUST-NAME +2 SLS-CUST-NO +2 PROD-DESC
```

SELECT Statement (Non-file SQL)

The non-file SQL SELECT statement allows CA-Easytrieve to retrieve rows without a file. This read-only method is retained from previous versions of CA-Easytrieve.

The SELECT statement identifies the rows and columns that are to be input to the JOB activity. Only one SELECT statement can be coded in each JOB activity, and it must be coded as the first statement in the JOB activity.

Syntax

```
                           { {*                 }
                [DISTINCT] { {expression        }
        SELECT  [        ] { {                  } +
                [ALL     ] { {table-name.*      }
                           { {correlation-name.*}

                                                         }
                           [  {expression        }     ] }
                           [, {table-name.*      } . ..] } +
                           [  {correlation-name.*}     ] }
                                                         }

         FROM table-name [correlation-name] +
              [,table-name [correlation-name] ...] +

          [WHERE search-condition] +

          [GROUP BY column-name        + ] +
          [         [, column-name ...]  ]

          [HAVING search-condition] +


           [                          { {*                 }          ]
           [             [DISTINCT]   { {expression        }          ]
           [UNION SELECT [        ]   { {                  } +        ]
           [             [ALL     ]   { {table-name.*      }          ]
           [                          { {correlation-name.*}          ]
           [                                                          ]
           [                                                      }   ]
           [                        [  {expression        }    ]  }   ]
           [                        [, {table-name.*      } ...]  } + ]
           [                        [  {correlation-name.*}    ]  }   ]
           [                                                      }   ]
           [                                                          ]
           [ FROM table-name [correlation-name] +                     ]
           [      [,table-name [correlation-name] ...] +              ] +
           [                                                          ]
           [  [WHERE search-condition] +                              ]
           [                                                          ]
           [  [GROUP BY column-name        + ] +                      ]
           [  [         [, column-name ...]  ]                        ]
           [                                                          ]
           [  [HAVING search-condition]                               ]

           [         {column-name} [ASC ]          ]
           [ORDER BY {           } [    ] +        ]
           [         {integer    } [DESC]          ]
           [                                       ]
           [         [  {column-name} [ASC ]    ]  ]
           [         [, {           } [    ] ...]  ] +
           [         [  {integer    } [DESC]    ]  ]

          INTO :host-variable [, :host-variable...]
```

Parameters

```
[DISTINCT]
[        ]
[ALL     ]
```

Specify DISTINCT to eliminate duplicate rows. ALL specifies that duplicate rows are not eliminated. ALL is the default.

```
{*                }
{expression       }
{                 }
{table-name.*     }
{correlation-name.*}
```

The above parameters are used to identify the columns to be retrieved from the specified table.

`FROM table-name [correlation-name]` *Table-name* specifies the table from which data is to be retrieved. *Correlation-name* can be used to specify an alternate qualifier for the *table-name* that immediately precedes it.

`[WHERE search-condition]` *Search-condition* is used to specify conditions for the retrieval of data. The *search-condition* is applied to the result of the FROM clause. Refer to your SQL vendor manuals for a description of the *search-condition*.

`[GROUP BY column-name]` GROUP BY is used to group data that is FETCHed into the file. *Column-name* must name a column in the *file-name*.

`[HAVING search-condition]` *Search-condition* is used to specify the data to be provided to the user. HAVING can be used to compare the results of all the returned data with a specific value in the data provided (such as the minimum or maximum value). Refer to your SQL vendor manuals for a description of the *search-condition*.

`[UNION...]` The UNION clause is used to include rows from another table.

```
[         {column-name} [ASC ] ]
[ORDER BY {           } [    ] ]
[         {integer    } [DESC] ]
```

ORDER BY returns the rows of the result table in the order of the values of the specified *column-names*. *Integer* references a column by its position in the result table rather than by a *column-name*. ASC returns the rows in ascending order and is the default. DESC returns the rows in descending order.

`INTO :host-variable [, :host-variable...]` INTO identifies where the column values are to be placed. The INTO clause must be the last clause coded on the SELECT statement.

Usage Notes

Code the SELECT statement immediately following the JOB INPUT SQL statement.

If this execution is for an SQL/DS system, a CONNECT statement is generated and executed by CA-Easytrieve. This means that the user does not need to include an SQL CONNECT statement when using CA-Easytrieve automatic processing. The user ID and password parameters are those that were specified in the USERID parameter of the PARM statement.

CA-Easytrieve checks the SQLCODE field following each execution of the select-clause. If the SQLCODE indicates an error, CA-Easytrieve issues an error message based on the SQL error and terminates execution. An SQLCODE value indicating end of data causes CA-Easytrieve to initiate end of input processing: the FINISH PROC (if any) executes, spooled reports are printed, and the current JOB activity ends. Refer to your SQL vendor's manuals for a description of SQL codes.

The SQL cursor that is automatically defined by a SELECT statement is closed following the JOB activity referencing it.

Example

The pseudo-code generated for automatic SQL processing is:

```
* IF SQL/DS
     SQL CONNECT :user-id IDENTIFIED BY :password
* END-IF
  SQL DECLARE cursor CURSOR FOR select clause
  SQL OPEN cursor
  DO WHILE SQLCODE NE 100
     SQL FETCH cursor INTO :host-variable-1  +
                        [, :host-variable-2...]
     process CA-Easytrieve code
  END-DO
  SQL CLOSE cursor
```

SELECT Statement (Report Selection)

A SELECT statement can be executed in a REPORT-INPUT procedure to select report input data. If the REPORT-INPUT procedure is not coded, all records selected with a PRINT statement are used in the report.

Syntax

```
SELECT
```

Usage Notes

SELECT only sets a switch to cause record selection at a later time. If you SELECT a record twice, it only appears once on the PRINTEDed report.

If coded, a REPORT-INPUT procedure is performed for each PRINT statement (report input). To cause the data to continue into report processing, you must execute a SELECT statement for the associated input data. In other words, input that does not get SELECTed is bypassed for continued processing. See the REPORT-INPUT Report Procedure for more information.

Example

```
REPORT-INPUT. PROC
    IF ZIP NE HOLD-ZIP
      HOLD-ZIP = ZIP
      SELECT
    END-IF
END-PROC
```

SELECT Statement (Sort Selection)

CA-Easytrieve supplies input records to your optional sort procedure one at a time. If a BEFORE procedure is used, a SELECT statement must be executed for each record that you want to sort.

Syntax

```
SELECT
```

Usage Notes

SELECT only sets a switch to cause record selection at a later time. If you SELECT a record twice, it only appears once on the SORTed file. If you SELECT a record and then issue a STOP, the record is not selected.

Example

The following example of a SORT activity shows an output file that contains only a reordered subset of the input file. The output file contains only those records for which the SELECT statement is executed.

```
FILE PERSNL FB(150 1800)
%PERSNL
FILE SORTWRK FB(150 1800) VIRTUAL
COPY PERSNL
SORT PERSNL TO SORTWRK USING  +
    (REGION, BRANCH, DEPT,    +
    NAME-LAST, NAME-FIRST)    +
    NAME MYSORT BEFORE SCREENER
*
SCREENER. PROC
   IF MARITAL-STAT = 'S' AND SEX = 1
      SELECT
   END-IF
END-PROC
```

SEQUENCE Statement

The SEQUENCE statement optionally specifies the order of a report or the values on a graph. You can order any report or graph based on the content of one or more fields.

Syntax

`SEQUENCE field-name [D] ...`

Parameters

`field-name` For reports, *field-name* identifies a field on which a report is ordered. You can specify multiple *field-names* for a report.

For graphs, *field-name* identifies a field on which graph values are sequenced. Only one *field-name* is permitted for a graph.

Field-name must be in an active file or W type working storage. Each field must be less than 256 bytes. The fields specified are used as sort keys processed in major to minor order.

Note: Varying length, K, and M fields cannot be specified on a SEQUENCE statement.

`[D]` An optional D following a *field-name* indicates that the field is sequenced into descending order. If you do not code D after a *field-name*, by default the field is sorted in ascending order.

Usage Notes

The fields used to SEQUENCE a report or graph do not have to be part of the printed report or displayed graph.

Examples

The following example illustrates using the SEQUENCE statement in a report declaration.

```
REPORT PERSNL-REPORT
  SEQUENCE REGION BRANCH PAY-NET D
  CONTROL REGION BRANCH
  TITLE 'PERSONNEL REPORT'
  LINE REGION BRANCH EMPNAME PAY-NET
```

The next example illustrates using the SEQUENCE statement in a graph declaration.

```
FILE PERSNL F(150)
%PERSNL
JOB INPUT PERSNL NAME DRAW-GRAPH
  DRAW GRAPH1
GRAPH GRAPH1 SUMMARY
  SEQUENCE BRANCH
  TITLE 'GROSS PAY BY BRANCH'
  VALUE BRANCH PAY-GROSS
```

The above graph declaration produces a pie graph that displays the sum of the gross pay for each branch sequenced by branch.

SET Statement

The SET statement allows you to dynamically change screen attributes and to control the display of screen errors.

Syntax

Format 1

```
SET field-name ERROR  +

  [ [     {attribute-name  } ] [ {'literal'  [   ] } ] ]
  [ [ATTR {                } ] [ {           [...] } ] ]
  [ [     {(attribute-list)} ] [ {field-name [   ] } ] ]
```

Format 2

```
                    {attribute-name  }
SET field-name ATTR {                }
                    {(attribute-list)}
```

Parameters

field-name *Field-name* specifies a field on a ROW statement in your screen declaration. *Field-name* can be indexed or subscripted. If the index or subscript of *field-name* is evaluated and is not on the screen, the SET statement is ignored.

ERROR Use ERROR to indicate that you want to flag *field-name* as being in error and to specify attributes or messages for *field-name*.

When you specify ERROR, the attributes or messages for *field-name* are determined by the hierarchy in the following table. The priority is from highest to lowest.

Statement/ Area	Attributes	Message
1. SET statement	ATTR parameter	*literal* or *field-name* parameter(s)
2. ROW statement	ERROR ATTR parameter	ERROR *'literal'* or *field-name* parameter(s)
3. DEFAULT statement	FIELD ERROR ATTR parameter	Default system message: Value entered is not allowed. Type an acceptable value.
4. Site Options Table	FIELD ERROR ATTR parameter	Default system message: Value entered is not allowed. Type an acceptable value.

Note: If you code SET ERROR without the ATTR or *'literal'/field-name* parameters, the attributes and messages are determined by the next statement or area in the above hierarchy.

```
[     {attribute-name  } ]
[ATTR {                } ]
[     {(attribute-list)} ]
```

ATTR specifies either a DECLAREd screen attribute name or one or more attribute keywords. See the ATTR Parameter for a list of attributes. See the DECLARE Statement for more information on DECLAREd screen attributes.

The following attributes are invalid for literals and system-defined read-only fields:

CURSOR MUSTENTER
NUMERIC TRIGGER
INVISIBLE ALARM
MUSTFILL

They are ignored if used, but CA-Easytrieve/Online issues a warning message during compilation.

SENDONLY and ASKIP are assumed for literals and system-defined read-only fields.

```
{'literal' }
{          }
{field-name}
```

Use *'literal'* to define the text you want displayed in the message. Use *field-name* to specify a field whose contents you want displayed as part of the message. A message can consist of a combination of *literals* and *field-names*.

The maximum length of a message is 130 characters. If the message exceeds the message area for the screen on which it is displayed, the message is truncated.

Usage Notes

You can code the SET statement in a screen procedure or in any procedure PERFORMed from a screen procedure, except for SCREEN TERMINATION. If coded in a SCREEN TERMINATION procedure or if coded in a procedure called from a SCREEN TERMINATION procedure, the SET statement is ignored at execution time.

The SET statement can be executed any number of times before displaying the screen. The last SET statement for *field-name* determines the attributes or messages for that field.

The attributes or error message established by a SET statement remain only for one iteration of the SCREEN activity. After the SCREEN is displayed, the attribute returns to its default as coded on the ROW statement. For changing field attributes until they are modified further, a declared attribute should be used. For an example, see Using Dynamic Screen Attributes in the *Programmer Guide*.

When multiple SET statements are coded for multiple *field-name*s before the next display of a screen, the *field-name* that is physically displayed first on the screen has its message displayed on the screen. All other *field-name*s only have their attributes displayed.

The SET statement overrides any ACTION messages defined in the MESSAGE statement, even if the MESSAGE statement is executed after all SET statements.

The attributes and messages specified on the SET statement are evaluated when the statement is executed. If the attributes or messages are variable, the value is saved and bound to the *field-name* when the SET statement is executed. If the variables are later modified, the attributes or messages are not changed when the screen is re-displayed.

If you code SET *field-name* ERROR without any other parameters, the attributes and messages for *field-name* are determined by the ROW statement. If the attributes and messages on the ROW statement are variable, the values displayed for the SET statement are the same as the values determined when the ROW statement is evaluated.

When you execute a RESHOW, REFRESH, or GOTO SCREEN statement after a SET statement, the attributes or messages specified in the SET statement are not affected.

Examples

In the following example, when the department number is not found in the table, the field is flagged in error.

```
ROW DEPT ERROR 'Department in error'
...
AFTER-SCREEN. PROC
  SEARCH DEPTBL WITH DEPT GIVING DEPT-DESC
  IF NOT DEPTBL
    SET DEPT ERROR
  END-IF
END-PROC
```

In the next example, when a user types a value greater than 50,000 into PAY-GROSS, PAY-GROSS is displayed in yellow, otherwise PAY-GROSS is displayed in turquoise.

```
ROW PAY-GROSS
...
AFTER-SCREEN. PROC
  IF PAY-GROSS > 50,000
    SET PAY-GROSS ATTR (YELLOW)
  ELSE
    SET PAY-GROSS ATTR (TURQ)
  END-IF
END-PROC
```

In the last example, five rows are displayed when the screen is displayed. Row 1 is displayed in BLUE and the cursor is positioned in AFIELD. Rows 2-5 are displayed in YELLOW.

```
REPEAT 5 TIMES VARYING SUB1 FROM 1
  ROW AFIELD(SUB1) ATTR (YELLOW)
END-REPEAT
...
BEFORE-SCREEN. PROC
  SET AFIELD(1) ATTR (BLUE CURSOR)
END-PROC
```

SKIP Statement

SKIP is a listing control statement that spaces the printer a designated number of lines before printing the next line of the statement listing.

Syntax

```
SKIP skip-amount
```

Parameters

`skip-amount` *Skip-amount* must be an unsigned integer.

Usage Notes

You can code a SKIP statement anywhere in CA-Easytrieve source code. SKIP must be on a record by itself. SKIP does not appear in the printed output. However, the requested blank line appears.

SORT Statement

The SORT statement defines and initiates an activity that sorts any file that can be processed sequentially. SORT sequences an input file in alphabetical or numerical order based on fields specified as keys.

Syntax

```
SORT input-file-name TO sorted-file-name +

  USING (sort-key-field-name [D] ...) +

  [        [ACTIVITY  ] [TERMINAL   ] ]
  [COMMIT ([           ] [           ])]  +
  [        [NOACTIVITY] [NOTERMINAL] ]

  [SIZE record-count] +

  [WORK number-of-work-data-sets] +

  [BEFORE proc-name] +

  [NAME sort-name]
```

Parameters

`input-file-name` *Input-file-name* is the name of the input file for the SORT activity.

Input-file-name must reference a FILE statement that defines a SEQUENTIAL, INDEXED, RELATIVE, or VFM file. The record length of *input-file-name* controls the length of records to be sorted, except when both files are fixed length. When this occurs, the length of the records is equal to that of *input-file-name* or *sorted-file-name*, whichever is shorter.

`TO` ***`sorted-file-name`*** *Sorted-file-name* designates the name of the output file of the sort activity. *Sorted-file-name* must reference a FILE statement that defines a SEQUENTIAL, INDEXED, RELATIVE, or VFM file.

If *sorted-file-name* is the same file name as *input-file-name*, the SORTed output is written over the input file.

`USING (`*`sort-key-field-name`*` [D] ...)` USING *(sort-key-field-name)* specifies key fields for sorting *input-file-name*.

You can code any number of fields up to the input limit of your installation's sort program. Up to 10 key fields can be used on the workstation. *Sort-key-field-name* can be any field less than 256 bytes long in the sort input file. (The only exceptions are variable length fields, which cannot be used as keys.) *Sort-key-field-name* cannot be a nullable field.

Code D to sort output in descending order. The default is ascending order.

Note: Varying length, K, and M fields cannot be specified as sort keys.

```
[        [ACTIVITY  ]  [TERMINAL  ] ]
[COMMIT ([          ]  [          ])]
[        [NOACTIVITY]  [NOTERMINAL] ]
```

Specify the COMMIT parameter to control the logical unit of work. COMMIT indicates when the activity commits recoverable work. Each commit point posts all updates, additions and deletions, and terminates holds. SQL cursors may or may not be closed, depending on the underlying database and the cursor definition.

Specify ACTIVITY to commit all recoverable work during the normal termination of the activity. Specify NOACTIVITY to tell CA-Easytrieve not to commit at the end of the activity. NOACTIVITY is the default.

Specify TERMINAL to commit all recoverable work during any terminal I/O operation. In CICS, this results in terminal I/O being performed in a pseudo-conversational mode. Specify NOTERMINAL to tell CA-Easytrieve not to commit during a terminal I/O. TERMINAL is the default.

If this activity is executed by an activity that has NOTERMINAL specified, this activity performs terminal I/O as if NOTERMINAL was specified.

See the *CA-Easytrieve Programmer Guide* for more information.

`[SIZE `*`record-count`*`]` Because CA-Easytrieve knows the number of records in files created by previous activities, it automatically supplies that information to the sort program. If the file was not created by a previous activity, you can enhance sort efficiency by supplying the approximate number of records as *record-count* on the optional SIZE parameter. *Record-count* must be an unsigned integer.

`[WORK` *`number-of-work-data-sets`*`]` Specify the number of work data sets used by the sort program. *Number-of-work-data-sets* must be one of the following:

- A zero – to indicate that DD statements are supplied
- A value from 1 to 31 – to indicate the number of work data sets that the sort program dynamically allocates.

This parameter overrides the number of work data sets set in the Site Options Table. WORK is ignored on the workstation.

`[BEFORE` *`proc-name`*`]` Optionally, specify *proc-name* to identify your procedure that pre-screens, modifies, and selects input records for the sort. See the SELECT (Sort Selection) Statement for more information.

If you do not specify BEFORE *proc-name*, CA-Easytrieve sorts all records in *input-file-name* and writes them to *sorted-file-name*.

`[NAME` *`sort-name`*`]` Optionally, specify *sort-name* to identify the SORT activity. *Sort-name* can:

- Be up to 128 characters in length
- Contain any character other than a delimiter
- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

The *sort-name* can be used to identify the sort in an EXECUTE statement.

Usage Notes

CA-Easytrieve supplies input records to your sort procedure one at a time. If a BEFORE *proc-name* procedure is used:

- You must execute a *SELECT* statement for each record that you want returned to the output file.
- A *SELECT*ed record is written only once, even if *SELECT*ed more than once in the procedure.
- Any record not *SELECT*ed does not get written to the sorted file.
- If the file being sorted is a variable length record file, the output file is generated with a record length equal to the maximum record length that is specified in the FILE statement.

SORT activities can be EXECUTEd by PROGRAM and SCREEN activities. If a PROGRAM activity is not coded, JOB and SORT activities are automatically executed sequentially until a SCREEN activity is encountered.

See the *CA-Easytrieve Programmer Guide* for more information on sorting files.

Example

In the following example, the output file contains all of the records of the input file sorted into ascending sequence by the values of fields REGION and BRANCH.

```
FILE PERSNL FB(150 1800)
%PERSNL
FILE SORTWRK FB(150 1800) VIRTUAL
COPY PERSNL
SORT PERSNL TO SORTWRK USING +
    (REGION, BRANCH) NAME MYSORT
```

SQL Statement

The SQL statement supports the SQL statements of the following database management systems:

- DB2
- SQL/DS
- CA-Datacom/DB SQL
- CA-IDMS SQL
- CA-Ingres
- Oracle
- SYBASE

Syntax

```
SQL native-sql-statement
```

Usage Notes

See the specific database management system manual information about syntax for native database statements. Listed below are the SQL statements currently supported by the SQL interface. See the *CA-Easytrieve Programmer Guide* for more information about coding native SQL statements.

DB2 SQL Statements:

ALTER
CLOSE cursor-name
COMMENT ON
COMMIT {work}

CONNECT
CREATE
DECLARE cursor-name {with hold}
DELETE {where current of cursor-name}
DROP
EXPLAIN
FETCH cursor-name
GRANT
INSERT
LABEL
LOCK
OPEN cursor-name
RELEASE
REVOKE
ROLLBACK {work}
SELECT INTO *(for static-only processing)
SET CONNECTION
SET CURRENT DEGREE
SET CURRENT PACKAGESET
SET CURRENT SQLID
SET host-variable
UPDATE {where current of cursor-name}

***Note:** Refer to SQLSYNTAX in the PARM statement.

SQL/DS SQL Statements:

ACQUIRE
ALTER
CLOSE cursor-name
COMMENT
COMMIT {work}
CONNECT userid
CONNECT TO database
CREATE
DECLARE CURSOR-NAME
DELETE {where current of cursor-name}
DROP
EXPLAIN
FETCH cursor-name
GRANT
INSERT
LABEL
LOCK
OPEN cursor-name
PUT
REVOKE
ROLLBACK {work}
UPDATE {where current of cursor-name}

CA-Datacom/DB SQL Statements:

ALTER
CLOSE cursor-name
COMMENT
COMMIT {work}
CREATE
DECLARE cursor-name
DELETE {where current of cursor-name}
DROP
FETCH cursor-name
GRANT
INSERT
LOCK
OPEN cursor-name
REVOKE
ROLLBACK {work}
SELECT INTO
UPDATE {where current of cursor-name}

CA-IDMS SQL Statements:

ALTER
CLOSE cursor-name
COMMIT {work} {continue} {release}
CONNECT TO dictionary-name
CREATE
DECLARE cursor-name
DELETE*
DROP
EXPLAIN
FETCH cursor-name
GRANT
INSERT
OPEN cursor-name
RELEASE
RESUME
REVOKE
ROLLBACK {work}
SUSPEND
UPDATE*

* **Note:** *WHERE CURRENT OF* cursor cannot be dynamically processed by the SQL interface for CA-IDMS. To perform SQL updates, you must code native SQL statements using a search *WHERE* clause.

CA-Ingres SQL Statements:

CLOSE cursor-name

COMMIT {work}
CONNECT
CREATE
DECLARE cursor-name
DELETE
DISCONNECT
DROP
FETCH cursor-name
INSERT
OPEN cursor-name
ROLLBACK {work}
UPDATE

Oracle SQL Statements:

CLOSE cursor-name
COMMIT {work}
CONNECT
CREATE
DECLARE cursor-name
DELETE
DISCONNECT
DROP
FETCH cursor-name
INSERT
OPEN cursor-name
ROLLBACK {work}
UPDATE

SYBASE SQL Statements:

CLOSE cursor-name
COMMIT {work}
CONNECT
CREATE
DECLARE cursor-name
DELETE
DISCONNECT
DROP
FETCH cursor-name
INSERT
OPEN cursor-name
ROLLBACK {work}
UPDATE
USE

SQL INCLUDE Statement

The CA-Easytrieve SQL INCLUDE statement indicates that SQL table information is to be used to generate CA-Easytrieve field definitions. It names the table and gives the location where the field definitions are generated.

Syntax

```
SQL INCLUDE +

 [(column ...)] +

 [         {starting-position} ]
 [         {* [+offset]      } ]
 [LOCATION {                 } ] +
 [         {W                } ]
 [         {S                } ]

 [HEADING] +

 [UPDATE] +

 [NULLABLE] +

 FROM [owner.] table
```

Parameters

[(*column* ...)] Specify a list of one or more column names for which field definitions are to be generated. The column name(s) must be enclosed within parentheses. If no column names are specified, all columns from the table are used.

```
[         {starting-position} ]
[         {* [+offset]      } ]
[LOCATION {                 } ]
[         {W                } ]
[         {S                } ]
```

Use this optional parameter to specify the location at which the field definitions are to be generated.

Starting-position specifies the starting position relative to position one of the record or file.

The * (asterisk) indicates that the field begins in the next available starting position (highest position assigned so far, plus 1). The optional +*offset* is an offset you want added to the * value. There must be at least one blank between the * and the optional +*offset*.

Coding W or S establishes a working storage field. W fields are spooled to report (work) files, S fields are not. W is the default location if the LOCATION parameter is not coded.

`[HEADING]` Optionally, code HEADING to cause remarks in the DBMS system catalog entry for a column to be copied into a HEADING parameter on the generated DEFINE statement for the column. This parameter is ignored for CA-Ingres.

`[UPDATE]` Code UPDATE to designate a modifiable column.

When a CA-Easytrieve SQL file does not contain the UPDATE parameter, only the specific columns defined with UPDATE can be modified with an UPDATE statement. If UPDATE is coded on the FILE statement, all columns in the file can be modified.

Note: You can only use UPDATE when the field definitions are generated for a CA-Easytrieve file.

`[NULLABLE]` Optionally code NULLABLE to cause default indicator fields to be defined for columns that contain NULL. The indicator field is defined as a 2 B 0 field preceding the field being defined. The default null indicator is automatically used by CA-Easytrieve whenever the associated column is referenced. You can override the use of the default null indicator by explicitly coding and referencing another indicator variable.

The indicator variable precedes the data portion of the field in storage. This field cannot be directly referenced. To check this indicator variable, you must use the IF NULL statement.

`FROM [`*`owner`*`.]` ***`table`*** FROM identifies the table definition to be defined to CA-Easytrieve. *Owner* is the optional 1 to 18-character alphanumeric qualifier, and *table* is the 1 to 32-character alphanumeric name. The period must be used as the qualification separator for owner-qualified tables.

Note: If the *owner* is not specified, the current authorization ID is used.

Usage Notes

When used, the SQL INCLUDE statement(s) must precede any other SQL or SELECT statements and must be coded in the library definition section of your CA-Easytrieve program.

The generated CA-Easytrieve field names are the same as the SQL column names. If a name matches a reserved word, the field definition is allowed, but all references to it must be qualified using any applicable qualification.

Mask information is not retrieved from the DBMS system catalog.

Group qualification structures of owner.table are defined prior to the first INCLUDEd definition. The fields are defined under the table entity, which is in turn under the owner level entity. This ensures that multiple tables with duplicate column names do not produce duplicate field names.

Fields with SQL data types that do not have equivalent CA-Easytrieve data types are defined as shown in the following table. Fields of DATE, TIME, TIMESTAMP, and BINARY cannot be used in arithmetic operations. Fields of FLOAT, DOUBLEPRECISION, REAL, and LONGINTEGER are defined as packed decimal fields. Non-zero FILE-STATUS and SQLCODE values are returned if the data is truncated.

SQL Data Type	CA-Easytrieve Data Type	Length	Decimals
DATE	Alphanumeric	10	
TIME	Alphanumeric	8	
TIMESTAMP	Alphanumeric	26	
BINARY	Alphanumeric	Length of SQL field	
FLOAT	Packed Numeric	10	3
DOUBLEPRECISION	Packed Numeric	10	3
REAL	Packed Numeric	10	3
LONGINTEGER	Packed Numeric	10	0

The DBMS system catalog must be referenced each time the program is compiled or interpreted. Therefore, to reduce catalog contention and to improve performance, you should always create link-edited programs.

Field Reference

One of the advantages of using the SQL INCLUDE interface is the ability to reference host-variables (CA-Easytrieve fields) using the group level TABLE definition.

When specifying the INTO clause on a native SQL FETCH or non-file SQL SELECT statement or the VALUES clause of the native SQL INSERT statement, the host variable TABLE definition can be substituted in place of coding all host-variables in the table.

If you require access to an indicator variable other than its use for NULL checking, you must define your own variable and reference it with its host-variable. For some DBMSs, the indicator variable is examined to detect truncation.

When the host-variable is a CA-Easytrieve group level definition of a table name, an array of type 2 B 0 should be specified immediately following the host-table-name-variable. The number of array elements should match the number of fields in the CA-Easytrieve table name definition. Array elements are matched one-to-one with the fields defined in the table name.

STOP Statement

The STOP statement terminates activities.

Syntax

```
STOP [EXECUTE]
```

Parameters

`[EXECUTE]` EXECUTE immediately terminates all CA-Easytrieve execution. STOP without EXECUTE terminates the current activity only.

Usage Notes

In CA-Easytrieve, activities with automatic file input automatically terminate when all input records have been processed. You can terminate activities prematurely, however, with a STOP statement. You must use STOP to terminate JOB activities without automatic file input (for example, JOB INPUT NULL).

Note: Use the EXIT statement to normally terminate SCREEN activities.

When used in a JOB activity, STOP completes all reports and executes a FINISH procedure, if coded. If you code STOP EXECUTE, all CA-Easytrieve activity procedures are immediately terminated. If STOP is coded in the START or FINISH procedure, the procedure is terminated.

When used in a SORT activity procedure, a STOP terminates the record selection process and executes the sort program. If you SELECTed a record, the record is not accepted.

When COMMIT ACTIVITY is specified for the activity, a STOP statement causes a COMMIT of all recoverable work. A STOP EXECUTE causes a ROLLBACK.

Examples

The following example illustrates STOP in a SORT activity to limit the number of records being sorted. In this example, only the first 50 records from PERSNL are sorted since the STOP statement simulates end-of-file on PERSNL.

```
FILE PERSNL  FB(150 1800)
%PERSNL
FILE SORTOUT FB(150 1800) VIRTUAL
COPY PERSNL
*
SORT PERSNL TO SORTOUT  +
    USING (PAY-GROSS D) +
    NAME MYSORT BEFORE SORT1-PROC
*
SORT1-PROC. PROC
    IF PERSNL:RECORD-COUNT GT 50
       STOP
    ELSE
       SELECT
    END-IF
END-PROC
```

Under certain circumstances, you might want to completely terminate all activities using a STOP EXECUTE statement, as in the next example.

```
FILE INVENT  FB(200 3200)
%INVMSTR
FILE SORTWRK F(200) VIRTUAL
COPY INVENT
*
JOB INPUT INVENT NAME MYPROG1 FINISH FINISH-PROC
   PRINT MYREPORT
*
FINISH-PROC. PROC
   IF RECORD-COUNT = 0
     DISPLAY 'INPUT FILE NOT AVAILABLE'
     DISPLAY 'HALTING EXECUTION...'
     STOP EXECUTE
   END-IF
END-PROC
*
REPORT MYREPORT
LINE PART-NUMBER PART-DESCRIPTION
*
SORT INVENT TO SORTWRK USING +
    (LOCATION-STATE,  +
     LOCATION-CITY) NAME MYSORT
*
JOB INPUT SORTWRK NAME MYPROG2
   PRINT MYREPORT
*
REPORT MYREPORT
LINE PART-NUMBER LOCATION-CITY LOCATION-STATE
```

SUM Statement

The SUM statement is a report definition statement that explicitly specifies the quantitative fields that are totaled for a control report.

Syntax

```
SUM field-name ...
```

Parameters

`field-name` *Field-name* is any quantitative field contained in an active file or W storage. You can specify multiple fields.

Usage Notes

Normally, CA-Easytrieve automatically totals all quantitative fields specified on LINE statements. The SUM statement overrides this process; only the fields specified on the SUM statement are totaled. The fields specified on a SUM statement do not have to be specified on a LINE statement. The SUM statement is only valid in a Control Report.

TERMINATION Report Procedure

A TERMINATION procedure is invoked at the end of the report. This procedure can be used to print report footing information, including control totals and distribution information.

Syntax

```
TERMINATION. PROC
```

Usage Notes

A TERMINATION procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

Example

The following is an example of report footing:

```
FILE FILE1
LAST-NAME  1  5 A
STATE      6  2 A
ZIP        8  5 N
PAY-NET    13 5 N 2
TOTAL-NET   S 8 N 2
JOB INPUT FILE1 NAME MYPROG
  TOTAL-NET = TOTAL-NET + PAY-NET
  PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65 +
  SUMMARY  SUMCTL DTLCOPY
  SEQUENCE STATE ZIP LAST-NAME
  CONTROL  STATE NEWPAGE ZIP
  TITLE 'REPORT FOR THE STATE OF' STATE
  LINE 01  LAST-NAME STATE ZIP PAY-NET
*
TERMINATION. PROC
  DISPLAY TITLE
  DISPLAY SKIP 5 TOTAL-NET 'IS THE Y-T-D COMPANY NET PAY'
  DISPLAY SKIP 5 'PLEASE ROUTE THIS REPORT TO CORPORATE OFFICERS'
END-PROC
```

TERMINATION Screen Procedure

A TERMINATION procedure is invoked once during the end of the screen activity.

Syntax

```
TERMINATION. PROC
```

Usage Notes

The TERMINATION procedure is performed when an EXIT action has been executed either from being assigned to a key or being executed in another screen procedure. It is used to perform actions that are to be executed only at the end of the activity.

If GOTO SCREEN or EXIT are executed in a TERMINATION procedure, the activity is stopped at that point. REFRESH and RESHOW are invalid in a TERMINATION activity.

A TERMINATION procedure must be delimited by an END-PROC statement. See the PROC Statement for more information.

TITLE Statement (Graphs)

The TITLE statement specifies the title to be displayed on the graph.

Syntax

```
      [[COL column-number] {'title-literal' }]
TITLE [[                 ] {                }] ...
      [[+offset          ] {title-field-name}]
```

Parameters

```
[COL column-number]
[+offset          ]
```

Use COL to display a title at a specific column (*column-number*) on the graph. Graphs are 80 columns wide when text is displayed.

Title items are separated by one space. Use +*offset* to add additional spaces between title items.

```
{'title-literal' }
{                }
{title-field-name}
```

'Title-literal' specifies a character string to be used for the graph title.

Title-field-name identifies an alphanumeric field to be used for the graph title. The *title-field-name* must be in an active file or W type working storage.

Example

The following program code produces a horizontal bar graph that displays the sum of the gross pay for each region with the title GROSS PAY BY REGION.

```
FILE PERSNL F(150)
%PERSNL
JOB INPUT PERSNL NAME DRAW-GRAPH
  DRAW GRAPH1
GRAPH VBAR GRAPH1 SUMMARY STYLE 'HBAR'
  SEQUENCE REGION
  TITLE COL 1 SYSDATE 'GROSS PAY BY REGION' COL 73 SYSTIME
  VALUE REGION PAY-GROSS
```

TITLE Statement (Reports)

One or more TITLE statements define an optional report title. The TITLE statement defines the title items and their position on the title line.

Syntax

```
                      {[             ] field-name}
                      {[#font-number]            }
TITLE [title-number] {[             ] 'literal' } ...
                      {+offset                   }
                      {-offset                   }
                      {COL column-number         }
```

Parameters

`[title-number]` *Title-number* specifies the position of the title line in the title area. *Title-number* must be from 1 to 99 (default is 1). You must specify title numbers in ascending order with no duplicates. The *title-number* of the first TITLE statement must be 1 or unspecified.

`[#font-number]` *(Mainframe and UNIX only)* #*Font-number* defines a font index. The value of #*font-number* identifies a font whose specifications are to be used for the next display item. You can only specify this option if the report has been associated with an extended reporting printer. #*Font-number* identifies the font number of a font defined for the associated extended reporting printer. If you do not code the font number, then the next display item uses the default font for the assigned extended reporting printer.

`{field-name}` *Field-name* specifies a field in any active file, working storage field, or system-defined field.

`{'literal'}` *'Literal'* specifies a character string for a title item. It must be either a numeric literal, hexadecimal literal, or an alphanumeric literal. Alphanumeric literals must be enclosed in single quotes.

By default, each title line is formatted as a list of title items that are separated by the number of spaces defined by the SPACE parameter of the REPORT statement. The +, -, and COL parameters modify this positioning.

Note: You must code at least one title item, specified by *field-name* or '*literal*', on each TITLE statement.

```
{+offset}
{       }
{-offset}
```

The space adjustment parameters, +*offset* or -*offset*, modify the normal spacing between title items. *Offset* is added to or subtracted from the SPACE parameter on the REPORT statement to get the absolute space between title items. The absolute space value can range from zero to any amount that still allows the title line to fit within the current LINESIZE value on the REPORT statement.

`{COL column-number}` The COL parameter specifies the print column number where the next title item is placed. The value of *column-number* has a valid range of 1 to 'nnn,' where 'nnn' cannot force the following title item beyond the end of the title line LINESIZE.

Each title line is centered within the title area of the report unless you specify NOADJUST.

When the report is associated with an extended reporting printer, an error results if two or more fields and/or literals overlap.

Usage Notes

CA-Easytrieve automatically positions the system date and current page count on title line one. This can be overridden by options on the REPORT statement (NODATE and NOPAGE).

TITLE Statement (Screens)

The TITLE statement is used to automatically center items on a screen.

Syntax

```
                   [[COL column-number] {field-name}
TITLE [row-number] [[                 ] {          }  +
                   [[+offset          ] {'literal' }

                   [     {attribute-name  } ] ]
                   [ATTR {                } ] ]  ...
                   [     {(attribute-list)} ] ]
```

Parameters

`[row-number]` Specify the *row-number* on which you want the TITLE to be displayed. If *row-number* is not specified, the next screen row is used for the title. The next screen row is not the highest row used, but the previously-specified row plus one. If no rows are previously specified, row one is used.

```
[COL column-number]
[+offset          ]
```

Use COL to display a title item at a specific column (*column-number*) on the screen.

Titles are separated by one space on a screen. Use +*offset* to add additional spaces between titles.

Note: A syntax error occurs when a TITLE overlays another screen item.

```
{field-name}
{          }
{'literal' }
```

Specify a *field-name* or a *'literal'* for the title. *Field-name* is the name of a field to be displayed as a title on the screen. *'Literal'* is an alphanumeric string to be displayed as a title on the screen.

```
[     {attribute-name  }
[ATTR {                }
[     {(attribute-list)}
```

Specify a DECLAREd screen attribute name or a list of attribute keywords. See the ATTR Parameter for a list of attributes. See the DECLARE Statement for procedures to DECLARE screen attributes.

Note: The following attributes are invalid for TITLEs:

CURSOR
NUMERIC
INVISIBLE
MUSTFILL
MUSTENTER
TRIGGER
ALARM

These attributes are ignored if used, but CA-Easytrieve issues a warning message during compilation. *SENDONLY* and *ASKIP* are assumed for *TITLE* items.

Usage Notes

TITLE items that are not located at a specific column (COL) are centered in the row based on the LINESIZE parameter of the SCREEN statement.

Example

The following TITLE statement:

```
TITLE 1 COL  1 'Date:' COL 7 SYSDATE 'Employee Master'   +
        COL 67 'Time:' COL 73 SYSTIME
```

produces:

```
Date: 12/31/89                   Employee Master                Time: 12:00:00
```

SYSDATE and SYSTIME are displayed starting in specific columns by using the COL parameter. 'Employee Master' is automatically centered.

TRANSFER Statement

The TRANSFER statement is used to transfer execution to a target program without returning to the invoking program.

Syntax

```
            {field-name    } [       {field-name} ]
   TRANSFER {              } [USING {           } ] [NOCLEAR]
            {'program-name'} [       {'literal' } ]
```

Parameters

```
{field-name    }
{              }
{'program-name'}
```

Specify the *field-name* that contains the name of the target program, or specify the name of the target program as a *'literal'* within single quotes. *Field-name* cannot be nullable.

```
[      {field-name} ]
[USING {          } ]
[      {'literal' } ]
```

Optionally, specify USING to pass a single parameter to the target program.

Specify the name of a field that contains the value to pass to the target program, or specify a *'literal'* to pass to the target program. *Field-name* cannot be nullable.

`[NOCLEAR]` Use NOCLEAR to specify that you do not want to clear the terminal screen when exiting a CA-Easytrieve program in CICS or on the workstation.

Usage Notes

The TRANSFER statement completely terminates the current CA-Easytrieve program and invokes the program specified by *program-name* or the program *field-name* using the linkage conventions of the operating system in which the program is executing. Issuing a TRANSFER statement is similar to issuing a STOP statement: reports are completed and a JOB FINISH procedure is executed (if coded).

The screen is automatically cleared when the current program terminates. In CICS and on the workstation, you can request that the screen remains displayed on the terminal by using the NOCLEAR parameter. In other environments, NOCLEAR is ignored and the screen is cleared and left in a ready mode.

Note: The target program inherits the execution environment of the program issuing the TRANSFER statement.

TRANSFER can be used to invoke any program written in any language that is supported by the operating system in which the program is executing; similarly, the program can issue any command supported by the operating system.

When the target program is another CA-Easytrieve program and you want to pass a parameter, you must specify the USING parameter on the target program's PROGRAM statement.

When transferring to another CA-Easytrieve program in a CICS pseudo-conversational environment, you must specify the TRANSID parameter on the target program's PARM statement.

Note: Use of the TRANSFER statement in interpretive execution causes the program execution to terminate.

See the *CA-Easytrieve Programmer Guide* for more information.

Example

```
CASE OPTION
  WHEN 'V'
    NEXT-PGM = 'VIEWCUST'
  WHEN 'E'
    NEXT-PGM = 'EDIT-CUST
  WHEN 'D'
    NEXT-PGM - 'DEL-CUST'
  WHEN 'A'
    NEXT-PGM = 'ADD-CUST'
END-CASE
TRANSFER NEXT-PGM USING EMP#
```

UPDATE Statement

The UPDATE statement is used to update a row from a CA-Easytrieve SQL file.

Syntax

```
UPDATE file-name
```

Parameters

file-name *File-name* is the name of a CA-Easytrieve SQL file.

Usage Notes

UPDATE issues an UPDATE WHERE CURRENT OF cursor.

When the file is defined with the UPDATE parameter, all defined columns are updated. Otherwise, only the columns that contain the UPDATE parameter are updated. See the SQL INCLUDE Statement or the DEFINE Statement.

Note: UPDATE WHERE CURRENT OF cursor cannot be dynamically processed by the SQL interface for CA-IDMS. To perform SQL updates, you must code native SQL statements using a searched update statement.

Example

The following example changes all employees in department 901 to department 921.

```
FILE PERSNL SQL PERSONNEL UPDATE
EMPNAME       *   20  A
WORKDEPT      *   2   P   0
EMPPHONE      *   3   P   0
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL
  SELECT FROM PERSNL WHERE WORKDEPT = 901 FOR UPDATE
  WORKDEPT = 921
  UPDATE PERSNL
```

VALUE Statement

The VALUE statement specifies the fields to be used to draw a graph.

Syntax

```
VALUE x-value {y-value [...]}
```

Parameters

`x-value` *X-value* specifies the field or literal to be used for drawing the horizontal axis of the graph. *X-value* must be a numeric field or literal if you are drawing an XY or SCATTER graph.

`{y-value [...] }` *Y-value* specifies the field and or literal to be used for drawing the vertical axis of the graph. Each field or literal must be numeric.

You can code a maximum of eight *y-values* to produce a multi-series graph. PIE charts are limited to one *y-value*.

Usage Notes

You can use a literal value to count occurrences. For example, the statement VALUE DEPT 1 counts the number of records within each department.

All graphs are automatically scaled; any data that is too long to be displayed is truncated.

S type working storage fields cannot be used in a VALUE statement.

All graphs are de-spooled before REPORTs/SYSPRINT.

The *y-value* for a PIE graph determines the size of the pie slice. The *x-value* for a PIE graph determines the category for the *y-value*. If SUMMARY is coded on the GRAPH statement, all *y-values* for each identical *x-value* are summed producing a slice that is the size of the sum of all of the y-values for this category.

For VBAR (vertical bar) and HBAR (horizontal bar) graphs, the *x-value* determines the category for the y-value. If SUMMARY is coded on the GRAPH statement, all *y-values* for each identical *x-value* are summed producing a vertical or horizontal bar that is the sum of all of the *y-values* for this category.

For LINE graphs, the *x-value* determines the category for the *y-value*. If SUMMARY is coded on the GRAPH statement, all *y-values* for each identical *x-value* are summed producing a data point that is the sum of all of the *y-values* for this category.

For XY and SCATTER graphs, the *y-value* determines position of the data point on the y-axis. The *x-value* determines the position of the data point on the x-axis. If SUMMARY is coded on the GRAPH statement, all *y-values* for each identical *x-value* are summed producing a data point that is the sum of all of the *y-values* for this *x-value*.

Example

The following code produces a stacked vertical bar graph that displays the sum of the gross pay and net pay for each region.

```
FILE PERSNL F(150)
%PERSNL
JOB INPUT PERSNL NAME DRAW-GRAPH
  DRAW GRAPH1
GRAPH VBAR GRAPH1 SUMMARY MODE('HIGH') TYPE('SVBAR')
  SEQUENCE REGION
  HEADING REGION ('Region')
  HEADING PAY-GROSS ('Gross' 'Pay')
  TITLE COL 1 SYSDATE 'GROSS/NET PAY BY REGION' COL 73 SYSTIME
  VALUE REGION PAY-GROSS PAY-NET
```

WRITE Statement

WRITE is used in the maintenance of SEQUENTIAL, INDEXED, and RELATIVE files (when allowed by the underlying access method). During random processing of these files, WRITE updates and deletes existing records and adds new records. Its syntax has two formats.

Syntax

Format 1

```
                      [UPDATE] [      {input-file-name   }]
     WRITE output-file-name [       ] [FROM {                   }][STATUS]
                      [ADD    ] [      {input-record-name}]
```

Format 2

```
     WRITE output-file-name DELETE [STATUS]
```

Parameters

output-file-name Specify the name of the SEQUENTIAL, INDEXED or RELATIVE file to be updated, added, or deleted. You must also code UPDATE on the FILE statement for *output-file-name*.

```
[UPDATE]
[ADD    ]
[DELETE]
```

Specify UPDATE, ADD, or DELETE to designate the type of file maintenance activity to be performed. UPDATE is the default.

For SEQUENTIAL files, only UPDATE is allowed. For RELATIVE files, only UPDATE and DELETE are allowed.

```
[      {input-file-name   }]
[FROM {                   }]
[      {input-record-name}]
```

Specify *input-file-name* or *input-record-name* to identify an alternate data source for file UPDATE and ADD operations. FROM is similar to coding a MOVE statement prior to a WRITE statement.

When *input-file-name* is specified, the current value of *output-file-name*:RECORD-LENGTH is the length of the output data. However, if the output file length is greater than the input file or record length, the excess storage is not initialized. Also, using the FROM parameter does not update the data area of the output file.

[STATUS] Specify the STATUS parameter whenever the possibility exists for an unsatisfactory completion of the input/output request.

STATUS checks input/output processing to see if it was performed properly. STATUS causes the file's FILE-STATUS field to be set with the appropriate return code. See Appendix A, "System-Defined Fields," to determine the meaning of the contents of FILE-STATUS. Normally, a zero or non-zero test is sufficient.

Note: FILE-STATUS is not defined if you do not specify a file type parameter on the FILE statement.

If you do not code STATUS and the operating system returns a non-zero status, CA-Easytrieve issues an appropriate diagnostic message.

Usage Notes

Format 1

Format 1 of the WRITE statement updates an existing record or adds a new record to the file. When updating, which is the default, the updated record is the current active record for the file.

Format 2

Format 2 of the WRITE statement deletes the current active record for the file.

Example

The following example illustrates the use of WRITE:

```
FILE PERSNL INDEXED UPDATE
%PERSNL
PROGRAM NAME MYPROG
   READ PERSNL KEY '05807' STATUS
   IF PERSNL:FILE-STATUS NE 0
     DISPLAY 'FILE-STATUS= ' PERSNL:FILE-STATUS
     DISPLAY 'UNSUCCESSFUL READ ON PERSNL FILE'
   ELSE
     DISPLAY HEX PERSNL
     MOVE '3125059599' TO TELEPHONE
     WRITE PERSNL UPDATE
     IF PERSNL:FILE-STATUS NE 0
       DISPLAY 'FILE-STATUS= ' PERSNL:FILE-STATUS
       DISPLAY 'UNSUCCESSFUL UPDATE ON PERSNL FILE'
     END-IF
   END-IF
```

Appendix

A System-Defined Files

Introduction

CA-Easytrieve automatically provides four categories of system-defined fields:

- General
- File-related
- Report-related
- Screen-related.

The fields in each of these categories are described next.

Note: Many system-defined fields are defined in the integer format. On the mainframe and in UNIX, the integer format is equivalent to binary format.

General Fields

SYSDATE

SYSDATE is a field that contains the system date at the start of CA-Easytrieve execution. SYSDATE is refreshed with the current date during each terminal I/O associated with a SCREEN statement. The DATE option set in the site options determines the format of the date. A slash (/) normally separates the month, day, and year components of the date (such as MM/DD/YY).

SYSDATE-LONG

SYSDATE-LONG is a field is similar to SYSDATE except that it includes the century in the system date at the start of CA-Easytrieve execution. An example of SYSDATE-LONG is MM/DD/YYYY.

SYSTIME

SYSTIME is a field that contains the system time at the start of CA-Easytrieve execution. SYSTIME is refreshed with the current time during each terminal I/O associated with a SCREEN statement. A colon (:) normally separates the data into hours, minutes, and seconds (such as HH:MM:SS).

SYSUSERID

SYSUSERID is a field identifying the terminal user.

In CICS, you must first log onto CICS because this field is updated from the EIB.

RETURN-CODE

RETURN-CODE is a field whose contents are returned to the MVS operating system in register 15 when CA-Easytrieve terminates. RETURN-CODE is initialized to zero, but you can set it to any value.

UIBFCTR

When processing an IMS/DLI database in a CICS environment, UIBFCTR contains the values from the UIBFCTR fields in the CICS UIB. See the *CICS Application Programmer's Reference* manual for a description of the UIBFCTR fields.

UIBDLTR

When processing an IMS/DLI database in a CICS environment, UIBDLTR contains the values from the UIBDLTR fields in the CICS UIB. See the *CICS Application Programmer's Reference* manual for a description of the UIBDLTR fields.

UIB-ADDRESS

When processing an IMS/DLI database in a CICS environment, UIB-ADDRESS contains the address of the CICS UIB. It only contains the UIB- ADDRESS following the execution of a Format 5 DL/I statement.

See the *CICS Application Programmer's Reference Manual* for a description of the UIB.

File Fields

CA-Easytrieve automatically provides the special data fields listed below for each of your files. These fields are stored as part of working storage but can be qualified by file-name. As working storage fields, they are not subject to invalid file reference errors.

RECORD-LENGTH

RECORD-LENGTH is a field used for all file types to determine or establish the length of the current data record. For variable-length records, this field contains only the length of the record's data. CA-Easytrieve automatically adjusts the field to account for the four-byte record-control-word and four-byte block-control-word. For variable-length files, assign the length of the record to the RECORD-LENGTH field before the PUT or WRITE statement is executed.

For SQL files, RECORD-LENGTH contains the sum of the maximum lengths of all fields in the file. For CA-IDMS and IMS/DLI files, RECORD-LENGTH contains the sum of the maximum lengths of all records in the file.

RECORD-COUNT

RECORD-COUNT is a read-only field that contains the number of logical input operations performed to the file.

For CA-IDMS and IMS/DLI files, only automatic input increments RECORD-COUNT.

FILE-STATUS

FILE-STATUS is a read-only field that contains the results of the most recent I/O operation on a file. FILE-STATUS is available when you code STATUS on the I/O statement. If you do not code STATUS, an appropriate error message is generated. The error message contains one of these codes.

For CA-IDMS files using automatic input, FILE-STATUS contains IDMSSTATUS. For IMS/DLI files, FILE-STATUS contains the status code from the PCB.

FILE-STATUS codes and their meanings are:

0000 Operation successful.

Explanation: This is not an error condition. It indicates that the last I/O operation was successful. No additional information is required.

0004 End of file.

Explanation: This is not an error condition. It indicates that the file position has been moved beyond the last record in the file.

Cause: This condition occurs following a GET statement when the current record is the last record in the file. It can occur for SEQUENTIAL, INDEXED, and RELATIVE files.

Following a GET PRIOR statement, this condition could also indicate the beginning of a file.

0008 Record with a duplicate alternate key exists.

Explanation: This is not an error condition. It indicates that the key of this record matches the key of the record that follows it in the sequential order of this file.

Cause: This condition can occur following a GET or READ statement for an INDEXED file that does not have unique keys.

Following a GET statement, this condition indicates that at least one more record with a matching key is waiting to be processed.

Following a READ statement, this condition indicates that there is at least one more record in the file with a matching key (a GET statement must be used to retrieve any remaining records).

In CICS/VS, MVS (batch and TSO), and CMS/OS, an INDEXED file can have non-unique keys if the associated data set is a VSAM PATH and the auxiliary index data set was defined with non-unique keys.

Note: There is no alternate or primary index on the workstation, in BTRIEVE, in UNIX, or in ISAM/VSAM, therefore, a status code of 8 is never encountered. However, if you move the application to the mainframe, you should test for this condition.

0012 Duplicate key.

Explanation: This error condition indicates that an attempt was made to store a record with a duplicate key, or there is a duplicate record for an alternate index with the Unique Key option.

Cause: This condition can occur following a PUT or WRITE ADD statement for an INDEXED file, or a PUT statement for a RELATIVE file.

For an INDEXED file, it indicates that the key of the record matches the key of a record already present in the file. For a RELATIVE file, it indicates that the slot designated by the relative record number already contains a record (the slot is not empty).

This condition can also occur following a WRITE UPDATE statement for a SEQUENTIAL or INDEXED file. It indicates that:

- There is at least one alternate index associated with this file.
- The alternate index was defined with the unique key and the upgrade option.
- The updated record caused a duplicate key condition to occur when the alternate index was updated.

0016 Record not found.

Explanation: This error condition indicates that the record designated by the KEY parameter is not found in the file.

Cause: This condition can occur following a READ or POINT statement for an INDEXED or RELATIVE file. For an INDEXED file, it indicates that no record in the file matches the key specified by the statement. For a RELATIVE file, this condition indicates that the slot designated by the relative record number is empty.

0020 Record locked.

Explanation: This error condition indicates that an attempt was made to access or update a record that has a lock placed on it by another process.

Cause: Condition is only possible on the workstation or in UNIX.

0024 Logical or physical error condition.

Explanation: This error condition indicates that a logical or physical error condition was detected by the access method routines used to access the file. The specific cause of the error is displayed in a runtime abend message. See Appendix A of the *CA-Easytrieve/Online User Guide* for a list of the feedback codes.

PATH-ID

For CA-IDMS and IMS/DLI files, PATH-ID is a field that contains the ID value of the lowest record retrieved in a path using the RETRIEVE statement. See the *CA-Easytrieve Programmer Guide* for more information.

IDMSCOM

IDMSCOM contains a set of fields defined for the CA-IDMS Communications Block. See the *CA-Easytrieve Programmer Guide* for more information.

SLC

SLC contains a set of fields defined for a logical record communications block.

See the *CA-Easytrieve Programmer Guide* for more information.

SQLCA

SQLCA contains a set of fields defined for the SQL Communications Block. See the *CA-Easytrieve Programmer Guide* for more information.

Report Fields

CA-Easytrieve automatically provides the special data fields listed below for your reports. These fields are stored as part of working storage and are read-only.

LINE-COUNT

LINE-COUNT is a field that contains the number of lines printed on the page.

LINE-NUMBER

LINE-NUMBER is a field that contains the number of the line being printed within the line group.

PAGE-COUNT

PAGE-COUNT is a field that contains the number of pages printed.

PAGE-NUMBER

PAGE-NUMBER is a field that contains the number of the page being printed.

TALLY

TALLY is a field that contains the number of detail records in a control break.

LEVEL

LEVEL is a field that indicates the control break level. See the CONTROL Statement.

BREAK-LEVEL

BREAK-LEVEL is a field that indicates the highest field in the break.

Screen Fields

CA-Easytrieve automatically provides the special data fields listed below for your screens. These fields are stored as part of working storage and are read-only.

KEY-PRESSED

KEY-PRESSED is a field that contains a value representing the most recent terminal key pressed by the terminal user.

CA-Easytrieve automatically defines symbolic names that correspond to values for the most common keys. Only keys with symbolic names can be used on a KEY statement.

Terminal Key	Symbolic Name	Constant Value
Unknown		0
Enter	ENTER	1
Clear	CLEAR	11
PA1 thru PA3	PA1 thru PA3	12 thru 14
PF1 thru PF24	F1 thru F24	21 thru 44
F1 thru F12	F1 thru F12	21 thru 32
Test Request		220
Op ID card Reader		222
Magnetic Slot Reader		223
Trigger Action		224
Structured Field		230
Clear Partition		231

Terminal Key	Symbolic Name	Constant Value
Read Partition		232
No Aid Generated		255

TERM-COLUMNS

TERM-COLUMNS is a field that contains the maximum number of columns the screen supports. You can test TERM-COLUMNS to execute a SCREEN activity designed specifically for the terminal being used.

TERM-ROWS

TERM-ROWS is a field containing the maximum number of rows the screen supports. You can test TERM-ROWS to execute a SCREEN activity designed specifically for the terminal being used.

TERM-NAME

TERM-NAME is a field containing the terminal identification. This field is set only in CICS environments.

Appendix

B Symbols and Reserved Words

Introduction

This appendix contains a list of CA-Easytrieve symbols and reserved words. The reserved words are listed in alphabetical order. Associated with each symbol is one or more references. The references describe the various ways you can use the symbol. An **R** in the column after the symbol indicates it is reserved.

Symbol References

Special Symbol	Reserved	Reference
.		Syntax delimiter (period) Macro parameter concatenation (period)
<		Conditional expression
<=		Conditional expression
(		Syntax delimiter (left parenthesis)
:		Syntax delimiter (colon)
+		Assignment Continuation of statements and words DISPLAY LINE TITLE
&		Macro variable prefix
*		Assignment Comment statement DEFINE
)		Syntax delimiter (right parenthesis)
¬<		Conditional expression POINT

Special Symbol	Reserved	Reference
¬>		Conditional expression
¬=		Conditional expression
-		Assignment Continuation of statements and words DISPLAY LINE TITLE
**	R	Reserved for future use
/		Assignment
'		Syntax delimiter (single quote)
%		Macro invocation
>		Conditional expression
>=		Conditional expression POINT
,		Syntax delimiter (comma)
=		Assignment Conditional expression POINT
@	R	Reserved for future use

Reserved Words

The following list includes all CA-Easytrieve reserved words.

ACCESS
AFTER-BREAK
AFTER-LINE
AFTER-SCREEN
AND
ATTR
BEFORE
BEFORE-BREAK
BEFORE-LINE
BEFORE-SCREEN
BREAK-LEVEL
BUSHU
BY
CALL
CASE
CHECKPOINT
CHKP
CHKP-STATUS
CLEAR
CLOSE
COL
COLOR
EOF
EQ
ERROR
EXECUTE
EXIT
EXTERNAL
F1, F2,..F24
FETCH
FILE
FILE-STATUS
FILL
FINAL
FIRST
FIRST-DUP
FOR
GE
GET
GO
GOTO
GQ
GR
GRAPH
LIST
LOW-VALUES
LQ
LS
LT
MASK
MATCHED
MEND
MESSAGE
MOVE
MSTART
NE
NEWPAGE
NOMASK
NOPRINT
NOT
NOTE
NOTITLE
NOVERIFY
NQ
NULL
OF
RESTART
RETRIEVE
RETURN-CODE
ROLLBACK
ROW
S
SCREEN
SEARCH
SECONDARY
SELECT
SEQUENCE
SET
SIZE
SKIP
SOKAKU
SORT
SQL
STOP
SUM
SYSDATE
SYSDATE-LONG
SYSIN

COMMIT
CONTROL
COPY
CURSOR
D
DECLARE
DEFAULT
DEFINE
DELETE
DENWA
DISPLAY
DLI
DO
DRAW
DUPLICATE
E
ELSE
ELSE-IF
END
END-CASE
END-DO
END-IF
END-PROC
ENDPAGE
END-REPEAT
ENDTABLE
ENTER
GT
HEADING
HEX
HIGH-VALUES
IDD
IDMS
IF
IN
INITIATION
INSERT
JOB
JUSTIFY
KANJI-DATE
KANJI-TIME
KANJI-DATE-LONG
KEY
KEY-PRESSED
KOKUGO
KUN
LAST-DUP
LE
LEVEL
LIKE
LINE
LINE-COUNT
LINE-NUMBER
LINK
OR
OTHERWISE
PA1..PA3
PAGE-COUNT
PAGE-NUMBER
PARM-REGISTER
PATH-ID
PATTERN
PERFORM
POINT
POS
PRIMARY
PRINT
PROC
PROCEDURE
PROGRAM
PUT
READ
RECORD
RECORD-COUNT
RECORD-LENGTH
REFRESH
RELEASE
RENUM
REPEAT
REPORT
REPORT-INPUT
RESHOW
SYSIPT
SYSLST
SYSPRINT
SYSSNAP
SYSTIME
SYSUSERID
TALLY
TERM-COLUMNS
TERM-NAME
TERM-ROWS
TERMINATION
TITLE
TO
TRANSFER
TRC
UNIQUE
UNTIL
UPDATE
UPPERCASE
VALUE
VERIFY
W
WHEN
WORK
WRITE
X
XRST

Appendix

C

Conversion from CA-Easytrieve Plus (Batch)

Introduction

This appendix contains information for sites converting from batch mainframe versions of CA-Easytrieve Plus (Releases 4.0 through 6.2) to the current versions of CA-Easytrieve/Online, CA-Easytrieve for UNIX, and CA-Easytrieve/Workstation.

This appendix also contains a table that helps you easily identify differences in CA-Easytrieve for different environments (mainframe, PC, and UNIX). These differences will be resolved in future versions of the CA-Easytrieve language. See Environmental Differences at the end of this appendix.

In this version of CA-Easytrieve, the syntax of several statements have been enhanced for future use. Wherever possible, older versions of the syntax are supported to allow your current file definitions and programs to operate.

This appendix provides information on the following topics:

- Differences between CA-Easytrieve versions
- Syntax from older versions still supported
- A list of features not supported in this version of CA-Easytrieve, but intended for future releases
- A table that identifies differences in CA-Easytrieve for different environments.

See Summary of Revisions in the “Overview” chapter for a list of enhancements in these versions from CA-Easytrieve Plus.

Differences Between Versions

You should be aware of the following differences between CA-Easytrieve versions.

BEFORE-LINE Procedure

You can use the BEFORE-LINE report procedure to modify detail line information in the new version. In previous versions, you could only use the REPORT-INPUT procedure to modify the detail line.

Bounds Checking

CA-Easytrieve now checks to ensure that your indexes and subscripts do not refer past the end of the field.

Conditional Expressions

Comparisons involving a VARYING alphanumeric field as the subject or object use the longer of the subject or object for the comparison and pad the shorter with spaces. Previous versions of CA-Easytrieve use the current length of the subject field as the length of the comparison.

DBCS Support

CA-Easytrieve/Online uses the CA-PSI Subsystems DBCS environment to define DBCS. CA-Easytrieve/Online supports only the IBM (Japanese) code system. Report processing is limited to EBCDIC and MIXED data formats for standard reports and DBCS data formats for extended reports. No spacing considerations are made for DBCS characters.

Note: CA-Easytrieve/Workstation and CA-Easytrieve for UNIX do not support DBCS.

DEFINE Statement

If you define the same field more than once with different attributes, an error occurs during compilation.

Fields whose storage location exceeds the file length are now flagged in error.

When you redefine a field that contains an OCCURS value, the field inherits the OCCURS value. This allows the field to be subscripted. If the field's length exceeds the redefined field's length, a warning message results when you define the field and subscripting is not allowed at reference. If a multi-dimensional array is defined, all occurrences of secondary dimensions must fit into a single occurrence of the corresponding primary dimension. A warning message results when you define the field and subscripting of secondary dimensions is not allowed at reference.

Fields referenced in activities in which no file I/O is specified now receive warning messages instead of error messages.

Varying length alphanumeric fields can now be modeled. If you use a varying length alphanumeric field as a model, the VARYING keyword is included in the length and type.

You cannot use the VALUE parameter for a varying length alphanumeric field unless the VARYING keyword precedes the VALUE parameter.

Extended Reporting

A program is provided to convert the CA-Easytrieve Plus (batch) extended reporting printer definition (EZTPXRPT) to the format used by CA-Easytrieve/Online (PSIXRPRT). The program (X020A) is in the sample PIELIB and includes sample JCL to be used for execution.

FILE Statement

The system-defined fields, FILE-STATUS, RECORD-LENGTH, and RECORD-COUNT are four-byte binary fields for files with SEQUENTIAL, INDEXED, RELATIVE or SQL specified on their FILE statement. Other file types use two-byte binary fields.

The FILE-STATUS field for files with SEQUENTIAL, INDEXED, or RELATIVE specified contains codes that are generic across access methods. Other file types use a FILE-STATUS obtained from the underlying access method.

Names

All entity names (files, fields, activities, reports, procedures, statement labels) can be up to 128 characters, rather than 40.

PARM Statement

Compile-and-go execution is not supported in the new version of CA-Easytrieve. Therefore, if neither the COMPILE, SYNTAX, nor LINK parameter of the PARM statement is specified, an object module is produced to SYSLIN|SYSLNK without a NAME|PHASE parameter.

REPORTs

The DISPLAY statement now ensures that printed output remains within the physical page size you specify.

Items on the title are now always refreshed with the current value of fields when produced. Older versions only refreshed title fields when printing titles as the result of a detail line.

Special-named break procedures now refer to the last detail record of the control group having the break. Older versions referred to the first record of the new control group.

The algorithm used to center ADJUSTed reports does not use the sum control tag literal. The report items are used to center the report, then the tag literal is placed on the report.

You can use the COL parameter on the TITLE statement without specifying the ADJUST parameter on the REPORT statement.

Reserved Words

There may be new reserved words in this version of CA-Easytrieve. See Appendix B for a complete list.

SYSDATE/SYSTIME

The SYSDATE and SYSTIME fields no longer use a space in place of a leading zero for month.

Supported Syntax

Following are discussions of syntax that is still supported but is not documented in the syntax diagrams contained in this manual. Refer to Batch CA-Easytrieve documentation for details. Some of the items in this chapter are ignored in this implementation, while others still function as before but are replaced by new syntax in this guide.

CALL Statement

The NR parameter is ignored.

DISPLAY Statement

The DISPLAY NEWPAGE function has been replaced by the DISPLAY TITLE and DISPLAY NOTITLE functions. The reason for this change is that previous versions of DISPLAY NEWPAGE did not consistently produce report titles and headings. For source compatibility, DISPLAY NEWPAGE is accepted and functions the same as DISPLAY TITLE.

END Statement

The END statement is effectively ignored because CARD input cannot be compiled with the program.

FILE Statement

The VS parameter is replaced with SEQUENTIAL, INDEXED, and RELATIVE. The underlying access method is determined at execution time. However, VS is still accepted as valid syntax.

The SQL SELECT parameter is replaced with the more powerful SQL file type. SQL SELECT is still supported.

PARM Statement

Current versions of CA-Easytrieve ignore the following:

LIST FILE|NOFILE
DEVICE
VFM DEVICE (except DISK|MEMORY)
SORT DIAG|NODIAG
SORT ERASE|NOERASE
SORT SYS
SORT TP|NOTP
SORT VIRTUAL|REAL
SORT WORK DA
PRESIZE
EXITSTR

The DBCSCODE parameter is replaced by the CODE parameter, but still supported as valid syntax.

Future Support

The current versions of CA-Easytrieve do not support the following items; they are intended for future releases.

VSE Batch Execution Support

VSE batch execution is not supported.

DBCS Support

Multiple code systems and DBCS extended reporting are not supported.

END Statement

Card input cannot reside with the source program.

FILE Statement

The DBCSCODE parameter of FILE is not supported.

JOB Statement

The ENVIRONMENT, CHECKPOINT, and RESTART parameters of the JOB statement are not supported.

Macros

VSE libraries are not supported as macro libraries.

PARM Statement

The ENVIRONMENT and RESTARTABLE parameters of the PARM statement are not supported.

Environmental Differences

The following table identifies differences in CA-Easytrieve for different environments (mainframe, PC, and UNIX). These differences will be resolved in future versions.

Feature	Mainframe	PC/DOS	UNIX
Screen Processing	X	X	
DECLARE Statement:			
PROGRAM Parameter	X		X
Graph Processing		X	
FILE Statement:			
CODE parameter	X	X	
SYSTEM parameter		X	
KEY parameter			X
LINK Statement:			
HOST parameter		X	
WAIT parameter		X	
NOENTER parameter		X	
PARM Statement:			
CODE parameter		X	
CALL parameter	X		X
Report Processing:			
CONTROLSKIP parameter	X		X
Double Byte Character Set (DBCS) Support	X		
CALL Statement:			
DYNAMIC execution	X		X
Conditional Expressions:			
Field Class Condition:			
BREAK parameter	X		X
HIGHEST-BREAK parameter	X		X

Index