

Advantage™ CA-Easytrieve®

Programmer Guide



Computer Associates®

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

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

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

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

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

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

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

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

© 2002 Computer Associates International, Inc. (CA)

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

Contents

Chapter 1: Overview

Introduction	1-1
About This Guide	1-1
Organization	1-2
Other CA-Easytrieve Publications	1-3
Related Publications	1-4
Documentation Conventions	1-5
Summary of Revisions	1-5
CA-Easytrieve/Online Enhancements from CA-Easytrieve Plus	1-5
CA-Easytrieve/Workstation Enhancements from CA-Easytrieve Plus/PC	1-6
CA-Easytrieve/Online 1.1 Enhancements	1-8
CA-Easytrieve/Online 1.4 Enhancements	1-9
CA-Easytrieve/Workstation 1.2 Enhancements	1-11
CA-Easytrieve 1.3 for UNIX Enhancements	1-11
CA-Easytrieve 1.4 for UNIX Enhancements	1-12

Chapter 2: Coding a CA-Easytrieve Program

Steps in the Application Development Process	2-1
Structured Programming Using CA-Easytrieve	2-1
CA-Easytrieve Program Sections	2-2
Environment Section	2-3
Library Section	2-3
Activity Section	2-3
Defining Files	2-5
Defining Fields	2-5
File Fields	2-6
Working Storage Fields	2-6
Signed/Unsigned Field Rules	2-7
Data Reference	2-8
Indexing	2-10

Subscripts	2-10
Varying Length Fields	2-10
Displaying Varying Length Fields	2-11
Assigning and Moving Varying Length Fields	2-11
Comparing Varying Length Fields	2-12
Declaring Screen Item Attributes	2-12
Declaring Input Edit Patterns	2-13
Declaring Subprogram Linkage	2-13
Literal and Data Formatting Rules	2-13
ASCII and EBCDIC Alphanumeric Literals	2-14
Hexadecimal Literals	2-14
UNIX Data Format	2-14
Format and Conversion Rules (Workstation Only)	2-14
Format Relationship Rules (Workstation Only)	2-15
Format and Conversion Rules (Mainframe Only)	2-18
Format Relationship Rules (Mainframe Only)	2-20
DBCS Format Literals	2-22
MIXED Format Literals	2-22
Controlling Program Flow	2-22
Activities	2-23
Program Flow	2-23
Screen Flow	2-24
Job Flow	2-24
Sort Flow	2-25
Units of Work/Commit Processing	2-25
Automatic Commit Processing	2-26
Controlled Commit Processing	2-26
Recoverable Resources	2-27
Decision and Branching Logic	2-29
Conditional Expressions	2-30
Double Byte Character Set Support	2-31
Assignments and Moves	2-31
Arithmetic Expressions	2-32
Operators	2-32
Parentheses	2-33
Evaluations	2-33
Assignment Statement	2-35
EBCDIC To DBCS Conversion (Mainframe Only)	2-36
Format 1 (Normal Assignment)	2-36
Examples	2-39
Format 2 (Logical Expression)	2-40

Example	2-41
MOVE Statement	2-41
MOVE LIKE Statement	2-42
Table Processing	2-42
Defining Tables	2-42
Searching Tables	2-44
Array Processing	2-44
Bounds Checking	2-45
Indexing	2-45
Single Dimension Arrays	2-45
Multiple Dimension Arrays	2-46
Subscripts	2-48
Subscripting a One-Dimensional Array	2-49
Subscripting a Two-Dimensional Array	2-49
Subscripting a Three-Dimensional Array	2-50
Segmented Data	2-51
Data Strings	2-53
Interprogram Linkage	2-54
CALL Statement on the Mainframe	2-55
Program Linking	2-55
Storage Management	2-56
Linkage (Register Usage) Conventions	2-57
Assembler Subprogram Linkage	2-57
COBOL Subprogram Linkage	2-58
Parameter Lists	2-59
Parameter List Format	2-59
Error Condition Handling	2-59
CALL Statement on the Workstation	2-60
Program Linking	2-60
Storage Management	2-60
Linkage Conventions	2-60
Error Condition Handling	2-65
CALL Statement in UNIX	2-65
Program Linking	2-65
Storage Management	2-66
Linkage Conventions	2-66
Error Condition Handling	2-70
LINK Statement	2-71
LINK vs. CALL	2-71
Commands Issued by the Child Program	2-71
USING Parameter	2-71

GIVING Parameter	2-72
Operating System Implementation	2-73
TRANSFER Statement	2-75
Commands Issued by the Target Program	2-75
Operating System Implementations	2-76
CA-Easytrieve/ESP Interactive Execution	2-77
Coding Efficient CA-Easytrieve Programs	2-77
Data Usage	2-77
Table Processing	2-78
Compiler Site and Program Options	2-78
Report Processing	2-79
Coding Programs That Run Under CICS	2-79

Chapter 3: File Processing

Overview	3-1
File Definition	3-2
Controlled vs. Automatic Processing	3-3
Data Buffering Mode	3-3
WORKAREA Parameter	3-3
Record Format	3-4
CARD, PUNCH, and VSAM	3-4
Record Address	3-5
STATUS Parameter	3-5
System-Defined File Fields	3-5
RECORD-LENGTH	3-5
RECORD-COUNT	3-6
FILE-STATUS	3-6
Error Conditions	3-6
Data Availability Tests	3-6
Opening Files	3-7
Closing Files	3-7
File Processing Modes	3-7
File Access Mode	3-8
Valid Syntax - FILE Statement	3-8
File Browse Mode	3-9
Hold/Release Processing	3-9
Workstation LANs	3-10
SEQUENTIAL Files	3-10
SEQUENTIAL File Processing Rules	3-10
SEQUENTIAL Input	3-11

Automatic Processing	3-11
Controlled Processing	3-11
CARD Input	3-11
SEQUENTIAL Output	3-12
Fixed-length File Creation	3-12
Variable-length File Creation	3-12
VSAM File Creation	3-12
PUNCH Files	3-13
Virtual File Manager	3-13
INDEXED Files	3-14
INDEXED Sequential Input	3-14
POINTing to a Specific Record	3-14
Skip-Sequential Processing	3-15
Random Input	3-15
Adding Records to an INDEXED File	3-15
Adding a Single Record	3-16
Mass-Sequential Record Insertion	3-16
File Creation	3-16
Deleting a Record	3-16
Updating a Record	3-17
RELATIVE Files	3-17
RELATIVE File Sequential Input	3-17
POINTing to a Specific Record	3-18
Skip-Sequential Processing	3-18
Random Input	3-18
Adding Records to a RELATIVE file	3-19
Record Addition	3-19
File Creation	3-19
Deleting a Record	3-19
Updating a Record	3-20
Sorting Files	3-20
SORT Activity Example	3-21
Sort Procedures	3-21
Sorting a Selected Portion of a File	3-22
Synchronized File Processing	3-22
Synchronized File Input	3-23
Example	3-23
Record Availability	3-24
Special IF Statements	3-25
MATCHED	3-25
File Existence	3-26

DUPLICATE, FIRST-DUP, and LAST-DUP	3-26
Updating a Master File	3-27
Single File Keyed Processing	3-27
PRINTER Files	3-28
Defining a PRINTER File	3-28
SYSPRINT	3-29
Workstation Files	3-29
Coding FILE Statements	3-29
SYSTEM Parameter	3-29
File Type	3-30
Record Format	3-30
Logical Record Length	3-31
File Code System	3-31
Allowed Field Types	3-31
Supported File Structures	3-32
Sequential	3-32
Indexed Sequential (FABS ISAM)	3-32
Relative (Random-access)	3-32
BTRIEVE	3-33
DBASE (xBASE)	3-33
LOTUS	3-34
CA-SuperCalc	3-35
Comma-Delimited	3-35
Host Mainframe	3-36
UNIX Files	3-37
File Type	3-37
Record Format	3-37
Logical Record Length	3-37
C-ISAM	3-38

Chapter 4: SQL Database Processing

Overview	4-1
Programming Methods	4-2
Native SQL Statements	4-2
Automatic Cursor Management	4-2
CA-Easytrieve SQL Statement Rules	4-3
Program Environment	4-3
Units of Work	4-4
PARM Statement Parameters	4-4
DB2	4-5

SQL/DS	4-6
CA-Datcom/PC	4-6
CA-Datcom/DB	4-7
CA-IDMS	4-7
CA-Ingres	4-7
ORACLE	4-7
Library Section Definition	4-7
SQL Catalog INCLUDE Facility	4-8
Processing NULLable Fields	4-8
Manual NULL Processing	4-9
SQL Data Types	4-9
Decimal Data Types	4-12
SQL Syntax Checking	4-12
System-Defined File Fields	4-13
RECORD-COUNT	4-13
RECORD-LENGTH	4-13
EOF Processing	4-13
SQL Communications Area Fields	4-13
Sample Database	4-16
Working Storage Definitions	4-16
CA-Easytrieve SQL Files	4-17
Processing Requirements	4-17
Operation	4-18
Input Processing	4-18
Automatic Processing	4-18
Controlled Processing	4-20
Update Processing	4-22
Controlled Processing	4-22
Automatic Processing	4-22
Deleting from an SQL File	4-23
Inserting an SQL Row	4-23
Automatic Retrieval without a File	4-23
Processing Requirements	4-23
Operation	4-24
Retrieving All Columns	4-24
Selected Columns	4-25
Multiple Tables	4-25
Native SQL Processing	4-25
Processing Requirements	4-25
Operation	4-26
Supported Commands	4-26

DB2	4-26
SQL/DS	4-26
CA-Datacom/PC	4-27
CA-Ingres	4-27
ORACLE	4-27
Unsupported SQL Commands	4-27
Retrieving All Columns	4-28
Reassign Departments	4-28
Update Phone Numbers	4-29
Using Table Name as Host Variable with Indicator Array	4-30
Data Types Supported on the Workstation	4-30

Chapter 5: CA-IDMS Database Processing

Introduction	5-1
CA-IDMS Interface	5-1
IDD Interface	5-1
CA-IDMS Functionality	5-2
CA-Easytrieve CA-IDMS Statements	5-2
Processing Overview	5-2
CA-IDMS Processing on the Workstation	5-3
Data Code System	5-3
Field Data Types	5-4
CA-IDMS Entity Names	5-4
Sample CA-IDMS Database	5-5
Field Definitions	5-5
Sample Logical Record	5-7
Logical Record Definition	5-7
IDD Interface	5-8
Program Name	5-9
Conforming IDD Item Descriptions to CA-Easytrieve Standards	5-9
Handling of Group Item Definition	5-9
Examples	5-10
IDMS Interface	5-10
Communications Block	5-11
Logical Record Communications Block	5-14
Using Logical and Element Records in Non-CA-IDMS Statements	5-14
Automatic Input	5-15
Sweep of an Area	5-16
Tickler File Control	5-16
Input Definition (Paths)	5-16

Automatic Input of Logical Records	5-17
WHERE Clause	5-17
Examples	5-17
Controlled Processing	5-21
IDMS Statement	5-22
Controlled Processing Examples	5-23

Chapter 6: IMS/DLI Database Processing

Introduction	6-1
Test Database	6-2
DBD Source Statements	6-2
PSB Source Statements	6-2
Test Database Structure	6-2
PCB and PSB Processing	6-3
PCB Specification and Access	6-3
PSB Specification	6-3
Status Information	6-3
Automatic Input	6-4
Sweep of Database	6-4
Tickler File Control	6-4
Input Definition (Paths)	6-5
Typical Path Examples	6-5
Tickler File Usage Example	6-6
Segment Selection Example	6-6
Path Identification Example	6-7
Complete Path Processing with Schedule and Terminate	6-8
Limiting Segment Retrieval	6-8
Root Segment Qualification Input Control	6-9
Conditional Segment Retrieval (Segment Pre-screening)	6-9
Controlled Processing	6-9
Complete Path Processing	6-10
Database Maintenance	6-11

Chapter 7: Report Processing

Overview	7-1
Basic Report Structure	7-2
PRINT Statement Processing	7-2
Work File Records	7-3
PRINT Workfile Processing	7-4

Report Formats	7-5
Standard Format	7-5
Label Format	7-6
REPORT Statement	7-7
Report Definition Structure	7-7
Report Definition Statements	7-7
System-Defined Fields	7-8
LINE-COUNT	7-8
LINE-NUMBER	7-8
PAGE-COUNT	7-8
PAGE-NUMBER	7-8
TALLY	7-8
LEVEL	7-8
BREAK-LEVEL	7-8
Standard Reports	7-9
Titles	7-9
Overriding Defaults	7-10
Examples	7-10
Headings	7-11
Line Group	7-12
Line Item Positioning	7-12
Special Positioning	7-13
Pre-printed Form Production	7-14
SPREAD Parameter	7-15
Label Reports	7-15
CONTROL Statement	7-16
Sequenced Reports	7-17
CONTROL Reports	7-17
Terminology	7-18
Data Reference	7-18
TALLY	7-19
LEVEL	7-19
BREAK-LEVEL	7-20
Control Report Contents	7-21
DTLCTL	7-23
SUMCTL	7-24
TAG	7-26
DTLCOPY	7-27
DTLCOPYALL	7-28
Control Field Values in Titles	7-28
Overflow of Total Values	7-29

Controlling Overflow	7-30
Summary File	7-32
SUMFILE Example	7-33
Report Procedures	7-33
Special-name Report Procedures	7-34
Coding Techniques	7-35
Field Reference	7-36
Static Working Storage	7-36
REPORT-INPUT	7-37
BEFORE-LINE and AFTER-LINE	7-38
BEFORE-BREAK	7-39
AFTER-BREAK	7-40
ENDPAGE	7-41
TERMINATION	7-42
Routing Printer Output	7-43
Reporting to the Terminal	7-44
Extended Reporting	7-46
Reporting Environment Example	7-47
Program Example	7-47
Printer Support	7-48
Printer Identification	7-49
Font Identification	7-49
CA-Easytrieve Printer Definitions	7-50
Page Mode Printers	7-50
Line Compatibility Mode Printers	7-54
XEROX Printers	7-55

Chapter 8: Screen Processing

Overview	8-1
Basic Structure	8-1
Screen Format	8-2
Title Area	8-2
Work Area	8-2
Message Area	8-2
Function Key Area	8-3
Screen Borders	8-3
Screen Example	8-3
SCREEN Statement	8-4
Screen Definition Statements	8-4
Screen Items	8-4

System-Defined Fields	8-6
KEY-PRESSED	8-6
TERM-COLUMNS	8-6
TERM-ROWS	8-7
TERM-NAME	8-7
SYSUSERID	8-7
Screen Title Area	8-7
Title Rules	8-7
Title Examples	8-8
Default Centering and Attributes	8-8
Explicit Locations and Attributes	8-9
Screen Work Area	8-9
Item Location	8-9
Location Examples	8-10
Attribute Examples	8-11
Formatting an Item for Display	8-11
Filling an Item for Display	8-12
Filling with Underscores	8-12
Filling with NULLs	8-12
Justifying a Field's Contents	8-13
Using Edit Masks for Display	8-13
Mask Example	8-14
Hexadecimal Mask Example	8-14
Automatic Editing of Input	8-14
UPPERCASE	8-15
MASK	8-15
PATTERN	8-16
Valid PATTERN Characters	8-17
Building Patterns	8-18
Character Sets	8-20
Internal Operation of Patterns	8-22
Additional Considerations	8-24
VALUE	8-25
Edit Error Messages	8-25
Cursor Placement	8-26
Cursor Placement Hierarchy	8-26
Repeating Rows of Data	8-27
A Simple REPEAT Example	8-27
Two-dimensional Arrays	8-28
Screen Message Area	8-28
Message Area Location	8-29

Message Attributes	8-29
Message Text	8-29
Screen Function Key Area	8-29
Location	8-30
Attributes	8-30
Screen Key Processing	8-30
3270 Display Station Keys	8-31
Screen Procedures	8-31
INITIATION	8-32
BEFORE-SCREEN	8-33
AFTER-SCREEN	8-33
TERMINATION	8-33
Programmer-Defined Procedures	8-33
Branch Actions	8-33
GOTO SCREEN	8-34
REFRESH	8-34
RESHOW	8-35
EXIT	8-36
CICS Pseudo-conversational Programs	8-36
Sending Messages	8-36
Using Message Levels	8-37
Determining the Cursor Location	8-38
Testing for Field Modification	8-38
Setting Errors	8-39
Commit Processing	8-40
SCREEN COMMIT Parameter	8-40
Conversational Processing Example	8-41
Pseudo-Conversational Processing Example	8-41
Concurrent Updates	8-43
SQL Processing Example	8-44
Sample Screen Applications	8-45
Editing Data and Setting Errors	8-46
Using Dynamic Screen Attributes	8-46
Using a Menu	8-47
Providing Help Screens	8-48
Field-specific Help	8-50
Windowed Screens	8-51
Action Bar Pull-Downs	8-52

Chapter 9: Graph Processing

Overview	9-1
Basic Structure	9-1
DRAW Statement Processing	9-2
Graph Format	9-2
Title Area	9-2
Work Area	9-2
Function Key Area	9-3
GRAPH Statement	9-3
Graph Definition Statements	9-3
Processing a Graph	9-4
Sample Graph Applications	9-5
Pie Chart	9-5
Vertical Bar Chart	9-6
Horizontal Bar Chart	9-7
Line Chart	9-8
Scatter Diagram	9-9

Chapter 10: System Services

CA-Easytrieve Macro Facility	10-1
Macro Invocation Statement	10-1
Syntax	10-1
Invoking Macros	10-2
Macro Library	10-2
Macro Files	10-2
Macro Library Security	10-3
CA-Panvalet	10-3
VSAM	10-3
Macro Definition	10-3
Macro Prototype Statement	10-4
Macro Body	10-5
Macro Termination Command	10-5
Macro Processing	10-5
Parameter Substitution	10-5
Examples	10-6
Parameter Substitution in a Macro	10-7
Instream Macros	10-7
Example	10-7

Glossary

Index

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% compatible) in the PC/DOS or OS/2 environment.

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

This *Programmer Guide* is designed for use by you, the CA-Easytrieve programmer. It assumes that you are familiar with the CA-Easytrieve language and understand basic data processing concepts.

The purpose of this guide is to help you:

- Apply CA-Easytrieve programs to various application tasks
- Create efficient CA-Easytrieve programs
- Analyze and modify existing CA-Easytrieve programs.

This guide is to be used with the following implementations of CA-Easytrieve:

- CA-Easytrieve/Online, version 1.4
- CA-Easytrieve/Workstation, version 1.2
- CA-Easytrieve in the UNIX environment, version 1.4.

Use of this guide, along with the *CA-Easytrieve Language Reference Guide*, provides you with the information needed to use CA-Easytrieve for all your programming needs. For a tutorial approach to learning the basics of CA-Easytrieve, see the *CA-Easytrieve Introduction to the Language Guide*.

Organization

This guide is divided into 10 chapters, a glossary, and an index:

Chapter 1, Overview Introduces you to CA-Easytrieve, this guide, and the related publications. Chapter 1 also lists the steps in the application development process, and discusses structured programming using CA-Easytrieve.

Chapter 2, Coding a CA-Easytrieve Program Provides you with CA-Easytrieve coding basics. Topics include data descriptions, program flow, table and array handling, assignments and moves, and subprogram conventions. Chapter 2 also includes tips on how to write programs in CICS.

Chapter 3, File Processing Describes the processing of sequential, indexed, and relative record files. Chapter 3 also describes printer file processing.

Chapter 4, SQL Database Processing Describes the two methods available for using CA-Easytrieve to retrieve and maintain data in an SQL database.

Chapter 5, CA-IDMS Database Processing Describes the optional processing facilities that interface with CA-IDMS databases and with the CA-IDMS Integrated Data Dictionary (IDD).

Chapter 6, IMS/DLI Database Processing Describes the optional processing facilities for information retrieval and maintenance of IMS/DL/I databases.

Chapter 7, Report Processing Describes how to produce printed reports using CA-Easytrieve.

Chapter 8, Screen Processing Describes how to display and receive information from an online terminal using CA-Easytrieve.

Chapter 9, Graph Processing Describes how to display and print graphs using CA-Easytrieve.

Chapter 10, System Services Describes how to use the CA-Easytrieve Macro Facility.

Glossary List of common CA-Easytrieve terms.

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

Other CA-Easytrieve Publications

In addition to this *CA-Easytrieve Programmer Guide*, Computer Associates provides the following CA-Easytrieve documentation:

Name	Contents
<i>CA-Easytrieve Language Reference Guide</i>	Describes the complete syntax of each CA-Easytrieve statement, organized in easy-to-find alphabetical order. Also provides lists of system-defined fields, symbols, and reserved words, as well as information for those sites converting to this version of CA-Easytrieve.
<i>CA-Easytrieve Introduction to the Language Guide</i>	Provides new users with the information they need to become productive quickly. Includes a tutorial and a format designed to make the material more interesting and easy to comprehend.
<i>CA-Easytrieve/Online User Guide</i>	How to compile, link-edit, and execute CA-Easytrieve programs on the mainframe. Also, how to compile and execute interactively using an editor and how to use the CA-Easytrieve/Online Screen and Report Painters.
<i>CA-Easytrieve/Online Administrator Guide</i>	Provides system administrators with the information needed to set up, customize, and administer a CA-Easytrieve/Online system.
<i>CA-Easytrieve/Online Installation Guide</i>	Describes installation of CA-Easytrieve/Online in all environments. The most current step-by-step procedures for installing CA-Easytrieve/Online in your environment(s) can be found on the product tape.
<i>CA-Easytrieve/Online CA-Activator Supplement</i>	How to install and maintain CA-Easytrieve/Online on your MVS system, using the Computer Associates SMP/E software interface called CA-Activator.
<i>CA-Easytrieve UNIX User Guide</i>	Helps compile, link-edit, and execute CA-Easytrieve programs in the UNIX environment from the

Name	Contents
	operating system command line.

Related Publications

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

- *CA-Easytrieve/Workstation User Guide*
- *CA-Easytrieve UNIX User Guide*
- *CA-Ingres SQL Reference Guide*
- *CA-PSI Subsystems DBCS Environment Guide*
- *CA-PSI Subsystems Reporting Environment Guide*
- *CA-Pan/SQL SQL Interface Installation Guide*
- *CA-Datacom/PC MS-DOS Database and System Administration Guide*
- *CA-Datacom/PC SQL Programming and Reference Guide*

These publications are also either referenced in this publication or are recommended reading:

- *SQL/Data System Application Programming for VSE (SH24-5018)*
- *SQL/Data System Application Programming for VM/SP (SH24-5068)*
- *CA-Datacom/PC SQL Programming and Reference Guide*
- *DATABASE 2 SQL Reference Manual (SC26-4380)*
- *Programmer's Reference Guide - COBOL*
- *IMS/DL/I Applications Programming Manual*
- *IMS/VS Installation Guide*
- *ORACLE SQL Reference Guide*
- *IBM Guide for New Users*
- *IBM CICS Application Programmer's Reference Manual*
- *IBM DLI 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 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.
<i>lowercase italics</i>	Lowercase italics represent variable information in statement 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 via 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 via 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	<p>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.</p> <p>Screens can now have a border built displayed around them, whether they are full-screen applications or windows.</p>
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	<p>These options provide the following capabilities:</p> <ul style="list-style-type: none"> ■ Automatic dropping of fractional results of calculations or assigns. ■ Automatic rounding off of fractional results of calculations ■ Truncation of digits during an assignment. <p>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.</p>
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	<p>A new Field Select window is available to display program fields for selection from other windows and lists.</p> <p>The Repeat Definition panel now automatically generates subscript fields on ROW statements.</p> <p>A new CAPS command provides a session override of the CA-Easytrieve/ESP Editor CAPS setting.</p>
CASE Statement	The CASE statement now supports variable length fields. If <i>field-name</i> is an alphanumeric literal, it no longer must be 254 or fewer bytes in length.

CA-Easytrieve/Online 1.4 Enhancements

CA-IDMS Processing	The CA-Easytrieve interface to CA-IDMS is now available in CA-Easytrieve/Online. This interface provides complete facilities for information retrieval and maintenance of CA-IDMS databases.
IMS/DL/I Processing	The CA-Easytrieve interface to IMS/DL/I is now available in CA-Easytrieve/Online. This interface provides complete facilities for information retrieval and maintenance of IMS/DL/I databases.
SQL Processing	<p>CA-Easytrieve SQL processing has been expanded to include interfaces to CA-Datacom and CA-IDMS SQL databases.</p> <p>A new PARM statement parameter, SQLSYNTAX, enables you to specify the level of syntax checking performed on SQL statements in your CA-Easytrieve program.</p>
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 <i>CA-Easytrieve Programmer Guide</i> .
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	<p>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.</p> <p>The DECLARE statement specifies how a particular subprogram is linked and overrides the CALL parameter on the PARM statement.</p>
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	<p>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.</p> <p>A new Options Table entry has been added called LONGDTE. This specifies the default date for reports.</p>

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-Datcom/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 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	<p>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.</p> <p>A new Options Table entry has been added called LONGDTE. This specifies the default date for reports.</p>
-------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

CA-Easytrieve 1.4 for UNIX Enhancements

	<p>CA-Easytrieve now operates on the IBM RS/6000 in AIX.</p>
DB2 Processing	<p>The CA-Easytrieve SQL Interface now supports DB2 in the UNIX environment.</p>
Extended Reporting	<p>A subset of the Extended Reporting feature is now available in UNIX. See the <i>CA-Easytrieve for UNIX User Guide</i> for details.</p>

Coding a CA-Easytrieve Program

Steps in the Application Development Process

The organization of this guide simulates the way an application program is developed. The basic application development process steps are:

1. Design the program:
 - Determine the task you want the program to accomplish.
 - Define the files necessary to read and write the information used by the program.
 - Fill in successive levels of the design until you have a program structure that accomplishes the task.
2. Code the program logic as designed:
 - Code and test the basic flow of your program before filling in lower levels of the design.

See the *CA-Easytrieve/Online User Guide*, the *CA-Easytrieve/Workstation User Guide*, or the *CA-Easytrieve UNIX User Guide* for instructions on compiling, link-editing, and executing your program.

Structured Programming Using CA-Easytrieve

The CA-Easytrieve language supports structured programming concepts by requiring you to use defined activities and special-named procedures. These activities and procedures help you create programs that are efficient, reliable, and maintainable.

CA-Easytrieve also allows you to practice structured programming concepts when you want to incorporate large sections of procedural code into your program.

CA-Easytrieve allows you to easily break a large program into manageable modules by using JOB, SCREEN, and SORT activities, and REPORT and SCREEN special-named procedures. You can code each module to perform a specific function for the program. Each specific function is then easily identified and maintained.

CA-Easytrieve allows you to create well-structured programs that can be read easily. To accomplish this, design your program with the following items in mind:

- Keep modules (procedures) small (small enough to fit on one page of the compile listing).
- Use meaningful comments wherever possible so that others can easily read and modify your program.
- Use the CA-Easytrieve control flow structures to make programs more readable and efficient. CA-Easytrieve provides two special GOTO statements that are very useful: GOTO JOB and GOTO SCREEN. These statements provide a well-defined method to instruct CA-Easytrieve to iterate the activity process.
- Use the following structured programming statements to control your program in a clear and logical way:

IF/ELSE/ELSE-IF
CASE
DO WHILE
DO UNTIL
EXECUTE
PERFORM

- Use consistent indentation that shows nesting and control flow. Indent statements that are enclosed in control flow structures, such as IFs and DOs. When nesting these control flow structures, indent an additional level. For example:

Poor Indentation

```
FILE FILEA
REGION 1 1 N
EMPNAME 17 20 A
JOB INPUT FILEA
IF REGION = 1
PRINT RPT
END-IF
REPORT RPT
LINE REGION EMPNAME
```

Good Indentation

```
FILE FILEA
  REGION 1 1 N
  EMPNAME 17 20 A
JOB INPUT FILEA
  IF REGION = 1
    PRINT RPT
  END-IF
REPORT RPT
  LINE REGION EMPNAME
```

CA-Easytrieve Program Sections

A CA-Easytrieve program consists of three main sections:

1. Environment section – optional
2. Library definition section – optional
3. Activity section(s) – at least one required.

Environment Section

The environment section enables you to customize the operating environment for the duration of a program's compilation and execution by overriding selected general standards for a CA-Easytrieve program.

Some of the standard CA-Easytrieve options affect the efficiency of a CA-Easytrieve program. There can be minor trade-offs between the automatic debugging tools provided by CA-Easytrieve and the efficiency of the program code.

For example, you can specify that CA-Easytrieve record the statement numbers of the statements being executed for display during an abnormal termination (FLOW). Use of this option, however, does have a minor impact on processing time. You can turn this option on or off in the environment section of each CA-Easytrieve program.

See Coding Efficient CA-Easytrieve Programs later in this chapter for details on how parameter settings affect your compilation.

Library Section

The library section describes the data to be processed by the program. It describes data files and their associated fields, as well as working storage requirements of a program. The library section is said to be optional because, on rare occasions, a program may not be doing any input or output of files. However, in most cases, use of the library definition section is required.

Processing time can be shortened by coding data definitions to avoid unnecessary data conversions. See Assignment and Moves later in this chapter for details on data conversions.

Activity Section

The executable statements that process your data are coded in one or more activity sections. Executable statements in CA-Easytrieve can be procedural statements or declarative statements.

The activity section is the only required section of your program. There are four types of activities: PROGRAM, SCREEN, JOB, and SORT.

- A PROGRAM activity is a simple top-down sequence of instructions. A PROGRAM activity can be used to conditionally execute the other types of activities using the EXECUTE statement.
- SCREEN activities define screen-oriented transactions. Data can be displayed to a terminal operator and received back into the program. Files can be read and updated. A SCREEN activity can EXECUTE a JOB or SORT activity to perform a special process such as printing a report.
- JOB activities read information from files, examine and manipulate data, write information to files, and initiate reports and graphs.
- SORT activities create sequenced or ordered files.

You can code one or more procedures (PROCs) at the end of each activity. Procedures are separate modules of program code you use to perform specific tasks.

REPORT subactivities are areas in a JOB activity where reports are described. You can code one or more REPORT subactivities after the PROCs (if any) at the end of each JOB activity. You must code any PROCs used within a REPORT subactivity (REPORT PROCs) immediately after the REPORT subactivity in which you use them.

GRAPH subactivities are areas in a JOB activity where graphs are described. One or more GRAPH subactivities can be coded after JOB procedures. You cannot code procedures for a GRAPH subactivity.

The following exhibit shows some CA-Easytrieve keywords and other items in the sections where they are usually located, and gives the general order of CA-Easytrieve statements within a program.

<pre> PARM ... </pre>	Environment Section
<pre> FILE ... DEFINE </pre>	Library Section
<pre> PROGRAM (statements) (program procedures) SCREEN (screen procedures) JOB (statements) (job procedures) REPORT (report procedures) GRAPH SORT (sort procedures) ... </pre>	Activity Section

Defining Files

Use the FILE statement to describe a file or a database. FILE statements must describe all files and databases that your program references. FILE statements are the first statements coded in the library section of a CA-Easytrieve program.

The FILE statement can differ greatly depending on the operating environment and the type of file being processed. See the "File Processing" chapter for more information.

Defining Fields

Use the DEFINE statement to define fields. The DEFINE statement specifies data fields within a record 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 section:

```

FILE PERSNL FB(150 1800)
  DEFINE EMP#      9   5   N
  DEFINE EMPNAME  17  20  A
  DEFINE EMP-COUNT W   4   N
*

```

Library

```

JOB INPUT PERSNL NAME MYPROG
EMP-COUNT = EMP-COUNT + 1
PRINT REPORT1
*
REPORT REPORT1
LINE EMP# EMPNAME EMP-COUNT
    
```

] Activity

The next example shows a field defined in the activity section:

```

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

] Library

] Activity

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

If the same field is defined more than once, subsequent definitions are ignored, that is, the first definition is used.

File Fields

File fields are normally defined immediately following the associated FILE statement in the library section of a CA-Easytrieve program. Their rules of usage are:

- CA-Easytrieve accepts an unlimited number of fields for each file (constrained by available memory).
- Field names must be unique within a file.
- You can define file fields anywhere in a CA-Easytrieve library or activity section, except within a REPORT subactivity or a SCREEN declaration.
- For more specific information about defining fields in a database file, see the appropriate SQL or CA-IDMS guide.

Working Storage Fields

You can specify two types of working storage fields: S (static) and W (work). Each type is used in a different way, particularly when used in reporting.

Fields defined as type S are stored in a static working storage area and are not copied onto report work files. All references to S fields in a report occur at the time the report is actually formatted and printed.

Fields defined as type W are copied onto the report work files at the time a PRINT statement is executed. A spooled report is not actually formatted and printed at the same time the PRINT is executed. Therefore, the value of a W field on a report is set at the time the report data is selected for printing, not at the time it is printed.

With this in mind, you should use S (static) working storage fields for:

- Temporary work fields for report procedures
- Line annotations controlled from report procedures
- Grand total values used to calculate percentages.

See the “Report Processing” chapter for examples of the use of W versus S fields.

Working storage fields are normally defined in the CA-Easytrieve library section. Their rules of usage are:

- CA-Easytrieve accepts an unlimited number of working storage fields (constrained by available memory).
- Working storage fields must be uniquely named within working storage.
- You can define working storage fields anywhere in a CA-Easytrieve library section, activity, or procedure.
- The sum of all working storage fields cannot exceed 32K (workstation only).

Signed/Unsigned Field Rules

Signed Fields

If you specify a numeric field with decimal positions (0 to 18), CA-Easytrieve considers it a signed (quantitative) field. The following rules apply to signed fields:

- For binary numbers, CA-Easytrieve takes the high-order (left-most) bit as the sign, regardless of field length. In any manipulation, CA-Easytrieve shifts the field and propagates the high-order bit. For example, a one-byte binary field containing a hexadecimal FF has the numeric value -1.
- For non-negative, zoned decimal numbers on the left side of an Assignment statement, CA-Easytrieve sets an F sign if EBCDIC, a 3 sign if ASCII. Otherwise, it manipulates the number in packed decimal format.
- Packed decimal numbers are manipulated in packed decimal format. Arithmetic operations that result in a positive result set a C sign.

- By definition, there is no sign in unsigned packed decimal numbers (U format). When you manipulate these numbers, CA-Easytrieve supplies an F sign.
- For integer numbers (workstation and UNIX only), the sign is manipulated as a 2's complement in native host format.
- For fixed point ASCII numbers (workstation only), the actual ASCII numeric data can reside anywhere within the field and can contain leading and/or trailing blanks or zeros. The sign must be the first non-blank character and all digits must follow without embedded blanks.

Unsigned Fields

If you specify a numeric field with no decimal positions, CA-Easytrieve considers that field unsigned (non-quantitative) and the following rules apply:

- For binary numbers, the magnitude of the number must fit within 31 bits or less. The NUMERIC test is not true for a four-byte binary field with the high-order bit on. The high-order bit contributes to the magnitude of numbers in fields of one-byte to three-byte lengths. For example, a one-byte binary field containing a hexadecimal FF has a numeric value of 255.
- Both zoned decimal and packed decimal fields follow the same rules. CA-Easytrieve packs all zoned decimal fields and handles them as packed fields. CA-Easytrieve uses the actual storage value in the field, but it is your responsibility to maintain a positive sign. An EBCDIC F sign or an ASCII 3 sign is placed in any unsigned field used on the left-hand side of an Assignment statement.
- An unsigned packed decimal field (U format) is always unsigned. When you manipulate the field, CA-Easytrieve supplies an F sign.
- For integer numbers, the high order bit is treated as part of the number increasing the positive magnitude to twice the signed magnitude plus one.
- For fixed point ASCII numbers, the actual ASCII numeric data can reside anywhere within the field and can contain leading and/or trailing blanks or zeros.

Data Reference

Every data reference (file or field) in your program must be unique. You can provide uniqueness in one of two ways:

- **Unique name** — A name is unique if no other file or work field has that name. For example, GROSS-PAY is unique if it appears as *field-name* in only one DEFINE statement (and has never been copied to another file with a COPY statement).

- **Qualification** – Qualification occurs when you prefix the optional qualifier *file-name:* to a field name. CA-Easytrieve requires the use of the qualifier whenever the field name alone cannot uniquely identify the data reference. The qualifier for file fields is the associated file name and/or record name. For working storage fields, the qualifier is the keyword WORK.

Default Qualification

Through default qualification, CA-Easytrieve attempts to determine which field you want to reference when a field name is not a unique name.

If you are in the library section, the current FILE (if one is coded) and the WORK file are searched for an occurrence of the field in question. If the field is not found in either the current file or the WORK file, or if the field occurs in both the current file and the WORK file, an error message is issued stating that additional qualification is required.

If you are in a SORT or JOB activity, and you code an INPUT *filename* on a JOB statement, the input file is searched for an occurrence of the field in question. For synchronized file processing, all of the files coded on the INPUT parameter of the JOB statement are checked for an occurrence of the field in question. If the field occurs in exactly one of the input files, that field is used. If the field occurs in more than one of the default files, an error message is issued stating that additional qualification is required.

If the field does not occur in any of the default files, each of the remaining files (including WORK) is searched for an occurrence of the field in question. If the field occurs in exactly one of the remaining files, that field is used. If the field occurs in more than one of the remaining files, an error message is issued stating that additional qualification is required.

If you do not specify an INPUT parameter on the JOB statement, a default input file is used. If the JOB activity immediately follows a SORT activity, the output from the SORT activity is the default input file. If the JOB activity occurs at any other point, the default input file is the first FILE coded that is not a TABLE file, a PUNCH file, or a PRINTER file. Once a default input file is selected, default qualification occurs as described above.

For PROGRAM, SCREEN, and JOB INPUT NULL activities, no default qualification occurs.

Indexing

Indexing is data reference that results from CA-Easytrieve deriving a displacement value to correspond to a particular occurrence in a field name defined with OCCURS. The formula for deriving the index value is: the number of the desired occurrence minus one, multiplied by the length of the occurring field element. For example, if an occurring field is defined as:

```
DEFINE MONTHWORD MONTH-TABLE 9 A OCCURS 12 INDEX MONTH-INDEX
```

MONTH-INDEX for the third occurrence is derived as follows:

```
MONTH-INDEX = (3 - 1) * 9
```

Or:

```
MONTH-INDEX = 18
```

See Array Processing later in this chapter for more information.

Subscripts

Subscripts are an alternate method available to select an individual element from an array. A subscript is a literal or field that contains the actual occurrence of the element you are referencing.

The use of subscripts removes the requirement of computing the index value; CA-Easytrieve does this automatically. See Array Processing later in this chapter for more information.

Varying Length Fields

The VARYING keyword on the DEFINE statement designates varying length alphanumeric fields. Varying length fields are often used in SQL databases. 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 which are internally defined as follows:

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

```
W 248 A for the data
```

When this field is referenced in your statements, you can designate the entire field including the length, by specifying FLDA. Or, you can specify only the length portion or only the data portion of the field. For the W field defined above:

FLDA:LENGTH references the binary portion only (bytes 1 and 2)

FLDA:DATA references alphanumeric portion only (bytes 3 - 250)

FLDA references the entire field (bytes 1 through 250)

When you reference the entire field, CA-Easytrieve automatically uses the length portion of the field when it acts on the field.

Displaying Varying Length Fields

The display “window” for varying length fields is based on the maximum length. However, the current value of the length portion determines how much of the data portion is actually displayed in the window.

Normally, the length portion of the field is not displayed. But when DISPLAY HEX is used, length as well as data are displayed. DISPLAY HEX displays length and the full data field in hexadecimal and character format. For example:

Statements:

```
DEFINE FLDA W 7 A VALUE 'ABCD' VARYING
JOB INPUT NULL NAME MYPROG
  DISPLAY FLDA
  DISPLAY HEX FLDA
STOP
```

Results:

```
ABCD
CHAR  ABCD
ZONE 00CCCC4
NUMB 0412340
```

Assigning and Moving Varying Length Fields

Assignments are based on the current length of the data and the rules of assignment. MOVES default to the current length of the data. MOVE SPACES moves blanks according to the maximum possible length of the varying length field. For example:

Statements:

```
DEFINE NULLSTRING W 10 A VARYING VALUE ''
DEFINE SENDVAR W 10 A VARYING VALUE '12345678'
DEFINE RECVM07 W 7 A VARYING
DEFINE RECVM10 W 10 A VARYING
JOB INPUT NULL NAME MYPROG
  RECVM10 = NULLSTRING . * ASSIGN NULL STRING TO VARYING
  DISPLAY '1. VALUE=' RECVM10 ' LENGTH=' RECVM10:LENGTH
  RECVM10 = SENDVAR . * ASSIGN 10 BYTE VARYING TO 10 BYTE VARYING
  DISPLAY '2. VALUE=' RECVM10 ' LENGTH=' RECVM10:LENGTH
  RECVM07 = SENDVAR . * ASSIGN 10 BYTE VARYING TO 7 BYTE VARYING
  DISPLAY '3. VALUE=' RECVM07 ' LENGTH=' RECVM07:LENGTH
  RECVM10 = RECVM07 . * ASSIGN 7 BYTE VARYING TO 10 BYTE VARYING
  DISPLAY '4. VALUE=' RECVM10 ' LENGTH=' RECVM10:LENGTH
  MOVE SPACES TO RECVM07 . * MOVE SPACES TO 7 BYTE VARYING
  DISPLAY '5. VALUE=' RECVM07 ' LENGTH=' RECVM07:LENGTH
STOP
```

Results:

1. VALUE=	LENGTH=	
2. VALUE=12345678	LENGTH=	8
3. VALUE=12345	LENGTH=	5
4. VALUE=12345	LENGTH=	5
5. VALUE=	LENGTH=	5

Note: If the sending field has length zero and the receiving field is a VARYING field, the receiving field has a length of zero. If the sending field has length zero and the receiving field is not a VARYING field, the receiving field is filled with the fill character (blank for assigned, blank or specified fill character for MOVE).

Comparing Varying Length Fields

Comparisons of varying length fields are based on the length of the data at the time of the comparison.

Declaring Screen Item Attributes

You can declare named screen attribute fields using the DECLARE statement. These declared attributes are different than ordinary fields that you DEFINE. Screen attributes contain information that controls the display of screen items such as their color and brightness.

Use of declared attributes provides the ability to dynamically change screen attributes during program execution, and saves you coding time when the set of attributes is used many times. Declared attributes can be used as follows:

- Use the DECLARE statement to name a set of screen attributes you want to use on multiple screen items.

For example, it is easier to use a declared attribute containing the INTENSE, BLUE, and MUSTFILL attributes on multiple items than to code the three attributes on each item in the screen. Declared attributes can be used in the ATTR parameter of the DEFAULT, TITLE, and ROW statements.
- You can use declared attributes to contain a dynamic set of attributes, that is, you can declare named attributes containing various sets of attributes. To do this, declare an empty named attribute to use on your ROW statements. Before displaying the screen, you can assign one of the declared attributes containing a specific set of attributes into the empty declared attribute. This allows you to dynamically set the screen attributes of items based on decisions made during execution.

See the “Screen Processing” chapter for details of the use of screen attributes.

Declaring Input Edit Patterns

The DECLARE statement can also be used to declare named input edit patterns. An input edit pattern allows you to specify a character sequence that describes the format of the data in the field.

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

These named input patterns can then be used in the PATTERN parameter of multiple items on ROW statements in a SCREEN activity. CA-Easytrieve automatically checks input data against the pattern and issues an appropriate error message to the terminal user if the data does not conform to its specified PATTERN.

Similar to declared attributes, using a declared pattern requires you to specify the PATTERN once. Declared patterns can then be used on multiple items on ROW statements.

See the “Screen Processing” chapter for a complete explanation of patterns.

Declaring Subprogram Linkage

The DECLARE statement can also be used to specify how you want to link a subprogram. Subprograms can be linked statically with your CA-Easytrieve program or dynamically loaded.

See Interprograms Linkage later in this chapter for details.

Literal and Data Formatting Rules

You can code the following types of literals in your CA-Easytrieve program:

- ASCII alphanumeric
- EBCDIC alphanumeric (mainframe and workstation only)
- Hexadecimal
- Double Byte Character Set (DBCS)
- MIXED format

CA-Easytrieve processes EBCDIC, ASCII, SBCS, MIXED, and DBCS data formats, but does not support all the possible relationships that can exist between these formats. The following topics describe each of the literals listed above, and explain the format and conversion rules and format relationship rules for the workstation and on the mainframe.

ASCII and EBCDIC Alphanumeric Literals

Alphanumeric literals are words enclosed within single quotes, and can be 254 characters long. An alphanumeric literal can contain ASCII and EBCDIC alphabetic characters A through Z, and numeric characters 0 through 9. Whenever an alphanumeric literal contains an embedded single quote, you must code two single quotes. For example, the literal O'KELLY is coded as:

```
'O' 'KELLY'
```

Note: ASCII is supported on the workstation and UNIX. EBCDIC is supported on the mainframe and workstation.

Hexadecimal Literals

Hexadecimal literals are words used to code values that contain characters not available on standard data entry keyboards. Prefix the hexadecimal literal with X' (the letter X and a single quote), and terminate it with a single quote. CA-Easytrieve compresses each pair of digits that you code within the single quotes into one character. CA-Easytrieve permits only the digits 0 through 9 and the letters A through F. The following hexadecimal literal defines two bytes of binary zeros:

```
X'0000'
```

UNIX Data Format

In the UNIX environment, CA-Easytrieve assumes all literals and alphanumeric data are ASCII and performs no conversion.

Format and Conversion Rules (Workstation Only)

During compilation, CA-Easytrieve converts all literals coded in the source program into the correct code system and data format (EBCDIC or ASCII) as dictated by the CA-Easytrieve statement in which they appear. To understand the process used to determine the correct code system and data format, it is important to identify the element of each CA-Easytrieve statement that is interpreted as the subject of that statement. The subject dictates the correct code system and format type.

Each CA-Easytrieve statement has a *subject element* whose code system and data format define the code system and data format of all the other elements that appear on that statement. The following table lists the subject element of those CA-Easytrieve statements that support the coding of literals. Those that do not have a subject element are also included. They are indicated by the words *not applicable*.

Statement	Subject Element
FILE <i>file-name</i>	<i>file-name</i>
DEFINE <i>field-name</i>	<i>field-name</i>
Assignment - <i>field-name</i> =	<i>field-name</i>
IF <i>field-name</i>	<i>field-name</i>
CASE <i>field-name</i>	<i>field-name</i>
DO WHILE UNTIL <i>field-name</i>	<i>field-name</i>
MOVE	<i>not applicable</i>
POINT <i>file-name</i>	<i>file-name</i>
CALL <i>program-name</i>	<i>field-name</i>
DISPLAY	<i>file-name(printer)</i>
REPORT <i>report-name</i>	<i>file-name(printer)</i>
HEADING	<i>file-name(printer)</i>
TITLE	<i>file-name(printer)</i>
LINE	<i>file-name(printer)</i>

Format Relationship Rules (Workstation Only)

CA-Easytrieve processes EBCDIC and ASCII data formats, but does not support all the possible relationships that can exist between these data formats. The following table defines the relationships CA-Easytrieve supports. CA-Easytrieve only supports the relationships defined in the table. Compilation errors occur if you specify any unsupported relationships in your CA-Easytrieve program.

If a conversion is necessary, the conversion column in the table indicates the additional processing that applies to get the object into the correct code system and applicable data format. The letter F means that CA-Easytrieve reformats the object to meet the requirements of the subject element. This category includes the reformatting of numeric data into the numeric format of the subject and also the reformatting from one data format (EBCDIC or ASCII) into the data format of the subject element. The letter C means that CA-Easytrieve performs a code system conversion when the code system of the data identified in the object does not match the code system of the subject element.

The workstation format relationships CA-Easytrieve supports are as follows:

Subject Data Format	Supported Object Data Format	Conversion
A - EBCDIC Alpha	EBCDIC Alpha field	
	Zoned Numeric field	F
	Packed field	F
	Unsigned field	F
	Binary field	F
	Integer field	F
	Hexadecimal literal	
	ASCII Fixed Point field	F C
	ASCII Alpha field	C
ASCII Alphanumeric literal	C	
A - ASCII Alpha	ASCII Alpha field	
	Zoned Numeric field	F
	Packed field	F
	Unsigned field	F
	Binary field	F
	Integer field	F
	Hexadecimal literal	
	ASCII Fixed Point field	F C
	EBCDIC Alpha field	C
EBCDIC Alphanumeric literal	C	
N - Zoned numeric	Zoned Numeric field	
	Packed field	F
	Unsigned field	F
	Binary field	F
	Integer field	F
	ASCII Fixed Point field	C F
P - Packed	Zoned Numeric field	F
	Packed field	
	Unsigned field	F
	Binary field	F
	Integer field	F
	ASCII Fixed Point field	C F
U - Unsigned Packed	Zoned Numeric field	F
	Packed field	F
	Unsigned field	
	Binary field	F
	Integer field	F
	ASCII Fixed Point field	C F
B - Binary	Zoned Numeric field	F
	Packed field	F
	Unsigned field	F

Subject Data Format	Supported Object Data Format	Conversion
	Binary field	
	Integer field	F
	ASCII Fixed Point field	C F
I - Integer	Zoned Numeric field	F
	Packed field	F
	Unsigned field	F
	Binary field	F
	Integer field	
	ASCII Fixed Point field	C F
F - ASCII fixed point	Zoned Numeric field	F
	Packed field	F
	Unsigned field	F
	Binary field	F
	Integer field	F
	ASCII Fixed Point field	

Format and Conversion Rules (Mainframe Only)

During compilation, CA-Easytrieve converts all literals coded in the source program into the correct DBCS code system and data format (SBCS, MIXED, or DBCS) as dictated by the CA-Easytrieve statement upon which they appear. To understand the process used to determine the correct code system and data format, it is important to identify the element of each CA-Easytrieve statement that is interpreted as the subject of that statement. The subject that dictates the correct code system and format type.

Literal Subject Elements (Mainframe)

Each CA-Easytrieve statement has a *subject element* whose DBCS code system and data format define the DBCS code system and data format of all the other elements that appear on that statement.

The following table lists the subject elements of those CA-Easytrieve statements that support the coding of literals. Those that do not have a subject element are also included. They are indicated by the words *not applicable*.

Statement	Subject Element
FILE <i>file-name</i>	<i>file-name</i>
DEFINE <i>field-name</i>	<i>field-name</i>
<i>Assignment</i> - <i>field-name</i>	<i>field-name</i>
IF <i>field-name</i>	<i>field-name</i>

Statement	Subject Element
DO WHILE <i>field-name</i>	<i>field-name</i>
RETRIEVE WHILE <i>field-name</i>	<i>field-name</i>
MOVE	<i>not applicable</i>
POINT <i>file-name</i>	<i>file-name</i>
CALL <i>program-name</i>	<i>not applicable</i>
DISPLAY	<i>file-name(printer)</i>
REPORT <i>report-name</i>	<i>file-name(printer)</i>
HEADING	<i>file-name(printer)</i>
TITLE (<i>report</i>)	<i>file-name(printer)</i>
LINE	<i>file-name(printer)</i>
SCREEN <i>screen-name</i>	<i>terminal</i>
DEFAULT	<i>not applicable</i>
KEY	<i>terminal</i>
TITLE (<i>screen</i>)	<i>terminal</i>
ROW/REPEAT	<i>terminal</i>

DBCS Code System and Data Format Literal Rules

Using the identified subject element of each CA-Easytrieve statement, the next table defines the rules for determining the DBCS code system and data format for a literal. If a literal is not in the required DBCS code system or data format, CA-Easytrieve converts the literal to the correct DBCS code system and data format during compilation.

In the following table, the code ASIS means that the data format (SBCS, MIXED, or DBCS) of the literal coded in the CA-Easytrieve source is retained by the CA-Easytrieve compiler (it is not converted).

Statement/ Keyword	Data Format of Literal	DBCS Code System of Literal
FILE - EXIT..USING	ASIS	PROCESSING
DEFINE - HEADING	ASIS	PROCESSING
IF/DO...WHILE	<i>field-name</i>	<i>field-name</i>
Assignment	<i>field-name</i>	<i>field-name</i>
POINT	ASIS	<i>file-name</i>

Statement/ Keyword	Data Format of Literal	DBCS Code System of Literal
HEADING	ASIS	<i>file-name(printer)</i>
TITLE (<i>report</i>)	ASIS	<i>file-name(printer)</i>
LINE	ASIS	<i>file-name(printer)</i>
KEY	ASIS	<i>terminal</i>
TITLE (<i>screen</i>)	ASIS	<i>terminal</i>
ROW	ASIS	<i>terminal</i>

Format Relationship Rules (Mainframe Only)

CA-Easytrieve processes SBCS, MIXED, and DBCS data formats, but does not support all the possible relationships that can exist between these data formats. The following table defines the relationships CA-Easytrieve supports. CA-Easytrieve does not support the relationships not defined in the table. Compilation errors occur if you specify them in your CA-Easytrieve program.

If a conversion is necessary, the conversion column in the table indicates the additional processing that applies to get the Object into the correct DBCS code system and applicable data format. The letter F means that CA-Easytrieve reformats the object to meet the requirements of the subject element. This category includes the reformatting of numeric data into the numeric format of the subject and also the reformatting from one data format (SBCS, MIXED, or DBCS) into the data format of the subject element.

The mainframe format relationships CA-Easytrieve supports are as follows:

Subject Element Data Format	Supported Object Data Format	Conversion
A - SBCS Alpha	SBCS Alphabetic field	
	SBCS Zoned Numeric field	F
	SBCS Packed field	F
	SBCS Unsigned Packed field	F
	SBCS Binary field	F
	SBCS Alphabetic Literal SBCS Hexadecimal Literal	
N - Zoned Numeric	SBCS Zoned Numeric field	
	SBCS Packed field	F
	SBCS Unsigned Packed field	F
	SBCS Binary field	F
	SBCS Numeric Literal	
P - Packed	SBCS Zoned Numeric field	F
	SBCS Packed field	
	SBCS Unsigned Packed field	F
	SBCS Binary field	F
	SBCS Numeric Literal	F
N - Unsigned Packed	SBCS Zoned Numeric field	F
	SBCS Packed field	
	SBCS Unsigned Packed field	F
	SBCS Binary field	F
	SBCS Numeric Literal	F
B - Binary	SBCS Zoned Numeric field	F
	SBCS Packed field	F
	SBCS Unsigned Packed field	F
	SBCS Binary field	
	SBCS Numeric Literal	F
M - Mixed	SBCS Alphabetic field	
	SBCS Zoned Numeric field	F
	SBCS Packed field	F
	SBCS Unsigned Packed field	F
	SBCS Binary field	F
	MIXED field	F C
	DBCS/Kanji field	F C
	SBCS Alphabetic Literal	
	SBCS Numeric Literal	F
	SBCS Hexadecimal Literal	
	DBCS Format Literal	F C
	MIXED Format Literal	C
K - DBCS/Kanji	SBCS Alphabetic field	
	SBCS Zoned Numeric field	F
	SBCS Packed field	F

Subject Element Data Format	Supported Object Data Format	Conversion
	SBCS Unsigned Packed field	F
	SBCS Binary field	F
	MIXED field	F
	DBCS/Kanji field	
	SBCS Alphabetic Literal	F
	SBCS Numeric Literal	F
	SBCS Hexadecimal Literal	
	DBCS Format Literal	
	MIXED Format Literal	F

DBCS Format Literals

DBCS format literals contain DBCS characters only. Enclose a DBCS format literal within apostrophes. A DBCS format literal can be 254 bytes long, including the shift codes. An example of a DBCS literal follows:

```
' [DBDBDBDBDBDB] '
```

The left bracket ([]) and right bracket (]) indicate shift-out and shift-in codes.

MIXED Format Literals

MIXED format literals are words containing both SBCS and DBCS characters. Enclose MIXED format literals within apostrophes. The presence of shift codes identifies DBCS subfields. Shift codes also identify the code system of that DBCS data. The word coded within the apostrophes (including the shift codes) cannot exceed 254 bytes in length. A MIXED literal is defined in the following example:

```
'EEEE [DBDBDB] '
```

The left bracket ([]) and right bracket (]) indicate shift-out and shift-in codes.

Controlling Program Flow

You control the flow of execution through your CA-Easytrieve program with the statements coded within your activities.

Activities

CA-Easytrieve activities resemble the steps of a batch job, but they are not constrained by Job Control Language (JCL) and associated operating system overhead. A CA-Easytrieve program consists of at least one of the four types of CA-Easytrieve activities: PROGRAM, SCREEN, JOB, and SORT.

A PROGRAM activity is optional, but if used, only one PROGRAM activity can be coded in a CA-Easytrieve program, and it must be coded before any other activities. PROGRAM activities can control the entire program. If used, the PROGRAM activity must EXECUTE the other types of activities when they are to be initiated. When not coded, there is an implied PROGRAM activity that initiates other activities as follows:

- JOB and SORT activities are executed sequentially until a SCREEN activity is detected.
- The SCREEN activity is executed.
- Any remaining activities must be executed by the first SCREEN activity. Automatic sequential execution does not proceed beyond the first SCREEN activity.

You can code one or more procedures (PROCs) at the end of each activity. Procedures are local to the activity after which they are coded. You cannot PERFORM procedures not associated with the activity in which the PERFORM is coded.

You can code one or more REPORT or GRAPH subactivities after the PROCs at the end of each JOB activity. You must code PROCs used within a REPORT subactivity immediately after the REPORT subactivity in which you use them.

Program Flow

The PROGRAM activity is a simple top-down execution of the statements contained in it. It is delimited by another activity in the source. PROGRAM activity execution stops when:

- The end of the activity is reached
- A STOP statement is executed in the PROGRAM activity
- A STOP EXECUTE or TRANSFER statement is executed anywhere in the program.

When a PROGRAM activity is coded, it is responsible for the execution of other activities in the program.

Screen Flow

The following shows the basic flow of a SCREEN activity. See the “Screen Processing” chapter for more information.

```

RESET working storage
[PERFORM INITIATION]
SCREEN ...
RESET working storage
  [PERFORM BEFORE-SCREEN]
    1. Build screen using program fields,
      pending messages, and cursor placement
    2. Send the screen
    3. Receive the screen
    4. Edit the input data
    5. Handle automatic actions
  [PERFORM AFTER-SCREEN]
GOTO SCREEN
RESET working storage
[PERFORM TERMINATION]
    
```

Job Flow

The following shows the relationship between JOB activity statements and shows implied statements attributed to JOB.

```

RESET working storage
[PERFORM start-proc]
JOB ...   retrieve automatic input
          IF EOF
            RESET working storage           Logic generated by JOB
            [PERFORM finish-proc]
            wrap-up REPORTS and GRAPHS
            return to invoking activity
          END-IF

          IF ...
            PERFORM proc-name
            PRINT report-name             JOB activity statements
            DRAW graph-name
            ...
          END-IF

          RESET working storage
          GOTO JOB                           Implied iteration at end
                                              of JOB statements

proc-name. PROC
  ...
END-PROC
GRAPH graph-name
REPORT report-name
  ...
JOB/SORT/SCREEN                             Optional procedures,
                                              graphs, and reports are
                                              placed at end of JOB
                                              statements

                                              Other activities
    
```

CA-Easytrieve processes input records one at a time. You can use any valid combination of CA-Easytrieve statements to examine and manipulate the input record. CA-Easytrieve repeats the processing activity until the input is exhausted or until you issue a STOP statement.

Sort Flow

The following exhibit illustrates the flow of a SORT activity.

Retrieve first record from input file (<i>file-a</i>)	Step 1 Logic generated by the SORT statement
DO WHILE NOT EOF <i>file-a</i>	
IF BEFORE was specified	Step 2 If BEFORE requested
IF RESET working storage fields specified	Step 3 Re-initialize RESET fields
reset all RESET working storage fields	Step 4 Perform the user's proc
PERFORM <i>proc-name</i>	Step 5 SELECT executed?
IF SELECT statement was executed	pass record to SORT
pass record to SORT	
END-IF	
ELSE	Step 6 No BEFORE proc, pass all to SORT
pass record to SORT	
END-IF	
Retrieve next record from input file (<i>file-a</i>)	Step 7 Get next record from input file
END-DO	
Perform SORT process (USING <i>fld1</i> , ...)	Step 8 Actually SORT the records
DO WHILE sorted records exist	Step 9 Write sorted records to output file
Write sorted record to output file (<i>file-b</i>)	
END-DO	
<i>proc-name</i> . PROC	Step 10 Optional user-written procedure is placed after the SORT
...	
SELECT	
...	
END-PROC	
JOB/SORT	

Units of Work/Commit Processing

To help you control the integrity of your files, databases, and other resources, CA-Easytrieve performs *commit processing*. Commit processing issues commands to the operating environment signifying the end of one *unit of work* and the start of another. These *commit points* provide a point at which updates are *committed* to the operating system. Changes that are not committed can be recovered or *rolled back*.

Commit points and rollbacks can be issued automatically by CA-Easytrieve or you can control this processing yourself.

Automatic Commit Processing

Each CA-Easytrieve activity can be considered a *logical unit of work*. For each activity statement, you can code a COMMIT parameter on the PROGRAM, JOB, SCREEN, and SORT statements to control the way CA-Easytrieve automatically issues commit points.

There are two times when CA-Easytrieve can automatically issue commit points:

- Use the ACTIVITY | NOACTIVITY subparameter to indicate whether CA-Easytrieve issues a commit point during the normal termination of an activity.
 - ACTIVITY - Tells CA-Easytrieve to commit during the activity's normal termination process. This is the default for PROGRAM activities, if not specified.
 - NOACTIVITY - Tells CA-Easytrieve not to commit during activity termination. This is the default for JOB, SCREEN, and SORT activities, if not specified.
- Use the TERMINAL | NOTERMINAL subparameter to indicate whether CA-Easytrieve issues a commit point during each terminal I/O operation performed in the activity.
 - TERMINAL - Tells CA-Easytrieve to commit during each terminal I/O. This includes terminal I/O for SCREEN activities and also for the Report Display Facility. In CICS, committing during terminal I/O runs your program pseudo-conversationally. This is the default, if not specified.
 - NOTERMINAL - Tells CA-Easytrieve not to commit during terminal I/O. In CICS, this runs your program conversationally.

Note: Once an activity with NOTERMINAL specified starts, all child activities execute with NOTERMINAL specified for them until the parent activity terminates.

Note: When CA-Easytrieve determines that a program has been linked to, the linked to program always behaves as if NOTERMINAL had been specified, that is, the child program always executes in fully-conversational mode. See Interprogram Linkage later in this chapter for complete details of the LINK statement.

When an activity terminates abnormally, CA-Easytrieve automatically issues a rollback to recover the updates made since the last commit point.

Note: If you execute a STOP EXECUTE statement in your activity, it is considered an abnormal termination.

Controlled Commit Processing

You can issue your own commit points and rollbacks as needed for your application by using the COMMIT and ROLLBACK statements. These commits and rollbacks are performed in addition to the commits and rollbacks automatically issued as a result of the COMMIT parameter on the activity statement.

Note: Controlled commits have no effect on whether programs run conversationally or pseudo-conversationally in CICS.

Recoverable Resources

A recoverable resource and the actual processing performed by the execution of commits and rollbacks is determined by the operating environment in which CA-Easytrieve is running. Each time a commit point is issued, CA-Easytrieve causes the following actions to happen:

- An SQL COMMIT that closes cursors is executed.
- An IDMS COMMIT or IDMS FINISH is executed. IDMS FINISH statements end run-units and are used at the end of activities or during terminal I/O. Following an IDMS FINISH statement, CA-Easytrieve does not automatically bind the run-unit or re-establish currencies.
- HOLDS that have been issued are released.
- Browsers of VSAM files are terminated. CA-Easytrieve can generally reposition the file for you. However, CA-Easytrieve cannot reposition an INDEXED file where the associated data set is a VSAM PATH and the auxiliary or secondary index data set was defined with non-unique keys.
- Browsers of UNIX C-ISAM files are terminated. As CA-Easytrieve only supports files with unique keys, it repositions the file for you.
- In CICS, printer spool files are closed. CA-Easytrieve automatically defers opening these files until they are used.

You must provide the necessary logic in your code to handle the above events.

CICS

In the CICS environment, the following resources are generally recoverable:

- SQL databases
- VSAM data sets specified as recoverable in their FCT entries
- CA-IDMS databases
- IMS/DLI databases.

In CICS, a request for a commit (either automatic or controlled) issues a CICS SYNCPOINT command. A rollback request issues a CICS SYNCPOINT ROLLBACK command. A CICS SYNCPOINT ROLLBACK command terminates the DLI PSB and database positioning is lost. CA-Easytrieve does not automatically re-schedule the PSB or reposition the DLI database.

When CA-IDMS is available, a request for a controlled commit issues an IDMS COMMIT command. An automatic commit issues an IDMS FINISH command. A rollback request issues an IDMS ROLLBACK command. Each time an IDMS FINISH command is issued, CA-IDMS ends the run-unit. You must provide the necessary logic in your program to bind a new run-unit and re-establish currencies as needed.

TSO and CMS

In TSO and CMS, SQL databases are the only recoverable resources. In TSO, CA-IDMS databases are also recoverable.

When SQL is available, a request for a commit (either automatic or controlled) issues an SQL COMMIT command. A rollback request issues an SQL ROLLBACK command.

Each time a commit point is issued, SQL closes all cursors. You must provide the necessary logic in your program to open and reposition the cursor as needed. Exceptions can exist for specific SQL databases that maintain cursor positioning across commits.

For DL/I files, issuing a commit point has no effect. In this case, DL/I CHECKPOINT and RESTART functions should be used to manage logical units of work. D/LI files do not lose positioning when a commit point is issued.

When CA-IDMS is available, a request for a controlled commit issues an IDMS COMMIT command. An automatic commit issues an IDMS FINISH command. A rollback request issues an IDMS ROLLBACK command. Each time an IDMS FINISH command is issued, CA-IDMS ends the run-unit. You must provide the necessary logic in your program to bind a new run-unit and re-establish currencies as needed.

Workstation

Generally, resources are not recoverable on the workstation. Updates are applied to the file immediately and cannot be rolled back. Commit processing does have an impact, however, on locks held on INDEXED file records when executing in a multi-user environment such as a Local Area Network (LAN). See Hold/Release Processing in the "File Processing" chapter for details of how and when CA-Easytrieve issues holds.

When SQL is available, a request for a commit (either automatic or controlled) issues an SQL COMMIT command. A rollback request issues an SQL ROLLBACK command. Each time a commit point is issued, SQL closes all cursors. You must provide the necessary logic in your program to open and reposition the cursor as needed.

When CA-IDMS is available, a request for a controlled commit issues an IDMS COMMIT command. An automatic commit issues an IDMS FINISH command. A rollback request issues an IDMS ROLLBACK command. Each time an IDMS FINISH command is issued, CA-IDMS ends the run-unit. You must provide the necessary logic in your program to bind a new run-unit and re-establish currencies as needed.

UNIX

In the UNIX environment, SQL, CA-IDMS, and C-ISAM are recoverable resources.

When SQL is available, a request for a commit (either automatic or controlled) issues an SQL COMMIT command. A rollback request issues an SQL ROLLBACK command. Each time a commit point is issued, SQL closes all cursors. You must provide the necessary logic in your program to open and reposition the cursor as needed.

When CA-IDMS is available, a request for a controlled commit issues an IDMS COMMIT command. An automatic commit issues an IDMS FINISH command. A rollback request issues an IDMS ROLLBACK command. Each time an IDMS FINISH command is issued, CA-IDMS ends the run-unit. You must provide the necessary logic in your program to bind a new run-unit and re-establish currencies as needed.

When C-ISAM is available, a request for a commit, either automatic or controlled, issues a call to *iscommit*. A rollback request issues a call to *isrollback*. Each time a commit point is issued, CA-Easytrieve closes and reopens all active C-ISAM files. Files are then repositioned upon the next browse operation.

Decision and Branching Logic

CA-Easytrieve uses certain statements to control the execution of your program by means of decision and branching logic. These statements can govern a program's execution flow depending on the truth value of the conditional expressions. The following statements are associated with decision and branching logic:

```
CASE  IF
DO    GOTO
EXECUTE  PERFORM
```

```
PROC STOP
EXIT REFRESH
RESHOW
```

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. The syntax of a conditional expression is:

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

CA-Easytrieve accepts seven different conditions. There are five simple conditions (having at most two operands) and two extended conditions (having potentially an unlimited number of operands):

Simple Conditions	Extended Conditions
Field Relational	Field Series
Field Class	File Relational
Field Bits	
File Presence	
Record Relational	

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

See the *CA-Easytrieve Language Reference Guide* for more information on each of these conditional expressions.

Double Byte Character Set Support

The following conditions provide support for DBCS and MIXED fields:

- Field Relational
- Field Series
- Field Class
- Field Bits

As with the data equations, when conversion from EBCDIC to DBCS format is part of the conditional expression, CA-Easytrieve converts the EBCDIC data using the technique defined below:

- CA-Easytrieve converts lower case EBCDIC values into the applicable DBCS Katakana characters.
- CA-Easytrieve converts other valid EBCDIC characters into their equivalent English values.
- CA-Easytrieve converts invalid EBCDIC values into DBCS spaces.

Combined Conditions

Any of these conditions, simple and extended, can be combined using the logical connectors AND or OR in any combination.

In the case of combined conditions, those connected by AND are evaluated first. The connected condition is true only if all of the conditions are true. The conditions connected by OR are then evaluated. The combined condition is true if any of the connected conditions are true. You can use parentheses to override the normal AND/OR relationships. The following table illustrates the results of combining conditions with AND, OR, and parentheses. The values x, y, and z represent any condition.

x	y	z	x OR y OR z	x AND y AND z	x OR y AND z	(x OR y) AND z
True	True	True	True	True	True	True
True	True	False	True	False	True	False
True	False	True	True	False	True	True
True	False	False	True	False	True	False
False	True	True	True	False	True	True
False	True	False	True	False	False	False
False	False	True	True	False	False	False
False	False	False	False	False	False	False

Assignments and Moves

The Assignment statement establishes the value of a field as a result of simple data movements, an arithmetic expression, or logical bit manipulation. If necessary, data is converted to the correct format depending on field type.

- An arithmetic expression produces a numeric value by adding, subtracting, multiplying, or dividing numeric quantities.
- The MOVE statement transfers, without conversion, character strings from one storage location to another.
- The MOVE LIKE statement copies fields with identical field names from one file to another. Assignments are generated for each field moved. Because assignments are used, the MOVE LIKE statement converts data to the correct format of the receiving field if necessary.

Arithmetic Expressions

To fully understand how an Assignment establishes the value of a field as a result of an arithmetic expression, you need to know how arithmetic expressions work within CA-Easytrieve. An arithmetic expression allows two or more numeric quantities to be combined to produce a single value. Arithmetic expressions can be used in Assignment statements and in field relational conditions.

Operators

The arithmetic operators used in CA-Easytrieve are:

Symbol	Operation
*	multiplication
/	division
+	addition
-	subtraction

All fields and literals in an arithmetic expression must be numeric. CA-Easytrieve follows the standard mathematical order of operations when computing arithmetic expressions: multiplication and division are performed before addition and subtraction, in order from left to right.

The following exhibit illustrates how CA-Easytrieve evaluates arithmetic expressions:

$$\begin{array}{l}
 11 + 5 * 8 - 48 / 16 + 4 \quad \text{Step 1} \\
 \quad \quad \quad \underbrace{\quad \quad \quad} \\
 11 + 40 - 48 / 16 + 4 \quad \text{Step 2} \\
 \quad \quad \quad \underbrace{\quad \quad \quad} \\
 11 + 40 - 3 + 4 \quad \text{Step 3} \\
 \underbrace{\quad \quad \quad} \\
 51 - 3 + 4 \quad \text{Step 4} \\
 \quad \quad \quad \underbrace{\quad \quad \quad} \\
 \quad \quad \quad 48 + 4 \quad \text{Step 5} \\
 \quad \quad \quad \underbrace{\quad \quad \quad} \\
 \quad \quad \quad 52
 \end{array}$$

Parentheses

You can use parentheses to override the normal order of evaluation. Any level of parenthesis nesting is allowed. CA-Easytrieve evaluates expressions within parentheses first, proceeding from innermost parenthesis level to the outermost.

The following exhibit illustrates how CA-Easytrieve evaluates parentheses found within arithmetic expressions:

$$\begin{array}{l}
 11 + 5 * ((8 - 48) / 16 + 4) \quad \text{Step 1} \\
 \quad \quad \quad \underbrace{\quad \quad \quad} \\
 11 + 5 * (-40 / 16 + 4) \quad \text{Step 2} \\
 \quad \quad \quad \underbrace{\quad \quad \quad} \\
 11 + 5 * (-2.5 + 4) \quad \text{Step 3} \\
 \quad \quad \quad \underbrace{\quad \quad \quad} \\
 11 + 5 * 1.5 \quad \text{Step 4} \\
 \underbrace{\quad \quad \quad} \\
 11 + 7.5 \quad \text{Step 5} \\
 \underbrace{\quad \quad \quad} \\
 18.5
 \end{array}$$

Evaluations

When evaluating an arithmetic expression, CA-Easytrieve maintains at most 30 decimal digits for each operation. During the calculation of:

$$\begin{array}{l}
 \text{field-name-1} \quad \{ = \} \quad \{ * \} \\
 \quad \quad \quad \{ \} \quad \{ / \} \\
 \quad \quad \quad \{ EQ \} \quad \{ \} \quad \text{value-1} \quad \{ \} \quad \text{value-2} \\
 \quad \quad \quad \quad \quad \quad \{ + \} \\
 \quad \quad \quad \quad \quad \quad \{ - \}
 \end{array}$$

the length and number of decimal places maintained during the calculation (intermediate results) is determined for each operation according to the rules shown in the following table.

If operation is:	The number of decimal places equals:
Addition or Subtraction	<p>Decimal places – The larger of the number of decimal places in value-1 or value-2.</p> <p>Length – The larger of the number of integer places in value-1 or value-2, plus the number of decimal places in result plus 1.</p>
Multiplication	<p>Decimal places – The sum of the number of decimal places in value-1 and value-2.</p> <p>Length – The sum of the length of value-1 and value-2.</p>
Division	<p>Decimal places – The larger of:</p> <ol style="list-style-type: none"> The number of decimal places in value-1 minus the number of decimal places in value-2. The number of decimal places in field-name-1 plus one. 4 decimal places. <p>Length – The number of integer places in value-1 plus the number of decimal places in the result.</p>

If the length of the intermediate result has more than 30 digits, CA-Easytrieve must truncate the excess digits. For addition, subtraction, and division, the excess digits are always truncated from the left side of the result.

For multiplication, however, CA-Easytrieve first attempts to do the truncation on the right side of the result. The minimum number of decimal places to be maintained in the result is the larger of:

- The number of decimal places in *field-name-1* plus one, or
- Four decimal places.

If the number of decimal places in the result is less than or equal to this minimum, no digits are truncated from the right side of the result. Otherwise, the number of digits truncated from the right is the smaller of:

- The number of excess digits

Or:

- The difference between the number of decimal places in the result and the minimum.

When truncation occurs on the right, both the length and number of decimal places in the result are reduced by the number of digits truncated. If there are still excess digits after right truncation, these excess digits are truncated from the left.

For example, assume that value-1 and value-2 both have a length of 18 digits and both have 4 decimal places. Then, according to the previous rules table, the result has a length of 36 digits and 8 decimal places. In this case, the number of excess digits is 6. Then, for various values of the number of decimal places in field-name-1, the result is truncated as shown in the next table:

Decimal places in field-name-1	Digits truncated on (right) (on left)		Decimal places in result
fewer than 2	6	0	4
2	5	1	4
3	4	2	4
4	3	3	5
5	2	4	6
6	1	5	7
more than 6	0	6	8

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:

Format 1 Syntax (Normal Assignment)

```

receive-field-name {= } {send-field-name }
                  { } {send-literal }
                  {EQ} {arithmetic-expression}
    
```

Format 2 Syntax (Logical Expression)

```

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

EBCDIC To DBCS Conversion (Mainframe Only)

When conversion from EBCDIC to Double Byte Character Set format is required for the Assignment statement, CA-Easytrieve converts the EBCDIC data using the techniques defined below:

- Converts lower case EBCDIC values into the applicable DBCS Katakana characters.
- Converts other valid EBCDIC characters into their equivalent English values.
- Converts invalid EBCDIC values into DBCS spaces.

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*. The rules of the statement are shown in the following table:

Receive-field-name (Left-hand side)	Send-field-name (Right-hand side)	Resulting Value
Alphanumeric field	Alphabetic field	Resulting value of <i>receive-field-name</i> is padded on right with spaces or truncated as necessary.
	Numeric field	Resulting value of <i>receive-field-name</i> is the zoned decimal equivalent of <i>send-field-name</i> with padding or truncation on left as necessary. Assignment of numeric fields to varying alphanumeric fields is not allowed.
	Alphanumeric or hexadecimal literal	Resulting value of <i>receive-field-name</i> is padded on right with spaces as necessary.
Alphanumeric or hexadecimal field	MIXED field	Each byte of <i>send-literal</i> is moved to <i>receive-field-name</i> unaltered. The resulting value of <i>receive-field-name</i> is padded on right with EBCDIC spaces.

Receive-field-name (Left-hand side)	Send-field-name (Right-hand side)	Resulting Value
Numeric field	Numeric field, literal, or arithmetic expression	Result is padded on left with zeros to fit the description of <i>receive-field-name</i> . If the value of the assignment is too large to be stored in <i>receive-field-name</i> , it is truncated as follows: <ul style="list-style-type: none"> ■ For Binary numbers (numbers expressed in Two's Complement form), the sign and high order bits are truncated from left as necessary, and the remaining left-most bit becomes the new sign. ■ For Zoned Decimal, Packed Decimal, and Unsigned Packed Decimal numbers (numbers expressed in Sign-Magnitude form), the high order digits are truncated from left as necessary. The result is truncated on right if the number of decimal places in <i>receive-field-name</i> is less than the right-hand side.
DECLARed attribute field	DECLARed attribute field	<i>Receive-field name</i> is replaced with the attributes contained in <i>send-field-name</i> .
Nullable field	NULL field	Indicator for <i>receive-field-name</i> set to 1 (indicates NULL).
	Not NULL field	The assignment works as usual and the indicator for <i>receive-field-name</i> is set to 0, indicating NOT NULL.
	Literal	Indicator for <i>receive-field-name</i> is set to 0.
	Arithmetic expression with any NULL operand	A runtime error occurs.
	Arithmetic expression in which all operands are NOT NULL	Indicator for <i>receive-field-name</i> is set to 0.
Not nullable field	NULL field	A runtime error occurs.
DBCS field	DBCS field	<i>Send-field-name</i> is converted into the DBCS code system of <i>receive-field-name</i> . The resulting value of <i>receive-field-name</i> is padded on right with DBCS spaces or truncated on right as necessary.
	MIXED field	Each EBCDIC byte of <i>send-field-name</i> is converted into its equivalent DBCS value. Any DBCS data identified by shift codes is converted to the DBCS code system of <i>receive-field-name</i> . The shift codes are then removed. The resulting value of <i>receive-field-name</i> is padded on right with DBCS spaces or truncated on right as necessary.

Receive-field-name (Left-hand side)	Send-field-name (Right-hand side)	Resulting Value
	Alphabetic field	Each byte of <i>send-field-name</i> is converted into its equivalent DBCS value and the resulting value is stored in <i>receive-field-name</i> . The resulting value of <i>receive-field-name</i> is padded on right with DBCS spaces or truncated on right as necessary.
	Numeric, packed, or binary field	Resulting value of <i>receive-field-name</i> is zoned decimal equivalent of <i>send-field-name</i> with each byte converted into the DBCS equivalent. Before the conversion, the result is padded on left with DBCS zeros, or truncated on left.
	DBCS literal	Resulting value of <i>receive-field-name</i> is padded on right with DBCS spaces or truncated on right as necessary.
DBCS Field	Alphanumeric or hexadecimal literal	Each byte of <i>send-literal</i> is converted into its equivalent DBCS value and the result is stored in <i>receive-field-name</i> . Resulting value of <i>receive-field-name</i> is padded on right with DBCS spaces or truncated on right as necessary.
MIXED field	DBCS field	<i>Send-field-name</i> is converted into the DBCS code system of <i>receive-field-name</i> . The shift codes defined for the code system of <i>receive-field-name</i> are added and the resulting value is padded on right with EBCDIC spaces or truncated on right as necessary. When truncation occurs, DBCS characters are not split. Truncation is to the nearest double byte.
	MIXED field	The EBCDIC data in <i>send-field-name</i> is moved unaltered to <i>receive-field-name</i> . The DBCS data identified by shift codes is converted to the DBCS code system of <i>receive-field-name</i> . The shift codes are also converted to meet the requirements of that code system. The resulting value of <i>receive-field-name</i> is padded on right with EBCDIC spaces or truncated on right as necessary. When truncation occurs within the DBCS portion of a field, DBCS characters are not split. Truncation is to the nearest double byte.
	Alphabetic field	Each byte of <i>send-field-name</i> is moved unaltered to <i>receive-field-name</i> . The resulting value of <i>receive-field-name</i> is padded on right with EBCDIC spaces or truncated on right as necessary.
	Numeric, packed, or binary field	Resulting value of <i>receive-field-name</i> is the zoned decimal equivalent of <i>send-field-name</i> with padding or truncation on left, if necessary.

Receive-field-name (Left-hand side)	Send-field-name (Right-hand side)	Resulting Value
	DBCS literal	<i>Send-field-name</i> is converted into the code system of <i>receive-field-name</i> and the correct shift codes are added. The result is padded on right with EBCDIC spaces.

Examples

The following examples of Format 1 of the Assignment statement illustrate its various rules:

**Format 1 (Normal Assignment, receive-field-name alphanumeric)
Statements:**

```
F1A W 4 A
F2A1 W 1 A VALUE 'A'
F2A2 W 6 A VALUE 'ABCDEF'
F2N1 W 2 N VALUE 12
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 earlier in this chapter.

**Format 1 (Normal Assignment, receive-field-name numeric)
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
$F1N = F2N1 + F2N2 + F2N3$ (1 + 2 + 3)	= 6.0
$F1N = F2N1 + F2N2 / F2N3$ (1 + 2 / 3) (1 + 0.6666)	= 1.6
$F1N = (F2N1 + F2N2) / F2N3$ ((1 + 2) / 3) (3 / 3)	= 1.0
$F1N = ((F2N1 / F2N2) * 100) + .5$ ((1 / 2) * 100) + .5 ((0.5 * 100) + .5) (50 + .5)	= 50.5

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.

- 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*.

Rules for Varying Length Fields

1. *Receive-field-name* and *send-field-name* must both be varying length fields or fixed length fields.
2. *Bit-mask-field-name* must be a fixed length field.
3. If *receive-field-name* is a varying length field, the length of its data portion must be equal to the length of the data portion of *send-field-length* and the length of *bit-mask-field-name* or *bit-mask-literal*.

Rules for Nullable Fields

The rules for nullable fields are shown in the following table:

Receive-field-name (Left-hand side)	Send-field-name (Right-hand side)	Resulting Value
------------------------------------------------	----------------------------------------------	------------------------

Nullable field	Nullable field but not NULL	The receiving field's indicator is set to 0, indicating NOT NULL.
	Not a nullable field	The receiving field's indicator is set to 0, indicating NOT NULL.
Not nullable field	NULL field	A runtime error occurs.

Example

The following example of Format 2 of the Assignment statement illustrates its various rules:

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
  
```

MOVE Statement

MOVE transfers characters from one storage location to another. It is used for moving data without conversion and for moving variable length data strings. The following table illustrates the rules of the MOVE statement regarding nullable fields:

Receive-field-name (Left-hand side)	Send-field-name (Right-hand side)	Resulting Value
Nullable field	<i>Send-field name</i> that is not nullable	Receiving field's indicator is set to 0, indicating NOT NULL.

	Literal	Receiving field's indicator is set to 0, indicating NOT NULL.
	<i>Send-field-name</i> that is nullable	The receiving field's indicator is set to -1 if the sending field is NULL, or if it is NOT NULL.
Not nullable field	NULL field	A runtime error occurs.

MOVE LIKE Statement

MOVE LIKE moves the contents of fields with identical names from one file to another. Data movement and conversion follow the rules of the Assignment statement.

Table Processing

A table is a collection of uniform data records that presents unique processing opportunities. All tables have two parts:

1. The *argument* uniquely identifies a table entry.
2. The *description* is the remainder of the table entry.

Some typical examples of table usage include organization structures, parts lists for assembly processes, and accounting chart-of-accounts.

The search of CA-Easytrieve table files is extremely efficient. Therefore, table use is recommended for applications that need to validate encoded data and/or retrieve code description.

Defining Tables

There are two types of tables that can be specified on the FILE statement:

1. ***Instream*** - (specified by the INSTREAM subparameter on the TABLE parameter) directs CA-Easytrieve to look for table data within the program immediately following the definition of the ARG and DESC fields for the file. This table is established at the time the program is compiled. Its size is limited only by the amount of available memory.
2. ***External*** - (INSTREAM is not specified) indicates that the table is located in a file external to the program. This file must be sequentially accessible. An external table is established just before use.

An external table can be:

- An existing file that is in ascending order by its search argument
- Created by specifying the name of the table as the TO *file-name* parameter in a SORT activity.

External tables that are also INDEXED files result in a random read to the file using the search argument as the key. This results in added efficiency.

All data needed to create small tables (to be processed by the SEARCH statement) can be entered instream along with CA-Easytrieve statements; that is, the table data can immediately follow the library definition statements for the table. The data is delimited by the ENDTABLE statement in the first eight positions of a record.

Instream table data is 80 characters per record and is unaffected by the SCANCOL options. All characters between the ARG and DESC definitions and the ENDTABLE delimiter are treated as data.

Note: An instream table can be retrieved from a macro file. However, the macro must contain the entire table definition (FILE statement through ENDTABLE).

The following illustrates a table-of-days definition:

```
FILE DAYTABL TABLE INSTREAM
  ARG 1 1 A. DESC 3 9 A
1 SUNDAY }
2 MONDAY }
...      } (instream data)
7 SATURDAY }
ENDTABLE }
```

The only way to modify an instream table is to recompile the program after supplying new table data. However, you can modify external tables without program change because CA-Easytrieve builds these tables dynamically prior to each use.

All tables must be sorted in ascending order by their search argument. No duplicate search arguments are allowed. Table sequence is validated as the table is created.

The only fields defined for table files are ARG (argument) and DESC (description). ARG defines the field used when searching the table. DESC defines the field which contains the desired information. The maximum length for an alphanumeric ARG or DESC field is 254 bytes.

The following illustrates a typical table file description. The resulting table provides descriptions of a hypothetical high school curriculum:

```
1011 ENGLISH I      }
1012 ENGLISH II    } records from
...      } CLASSES file
...      }
```

```
9712     HOME ECONOMICS }  
-----  
FILE    CLASSES  TABLE (150)...  
ARG 1   4  A.          DESC 10 40 A |
```

Searching Tables

The SEARCH statement provides access to table information. You can code SEARCH statements any place within a PROGRAM, SCREEN, or JOB activity, and issue any number of SEARCHes against any number of tables. To test the success of the SEARCH, use the file presence test: IF [NOT] *file-name*.

The following 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  
  
Result:  
ENGLISH II
```

Array Processing

An array is a series of consecutive memory locations in one or more dimensions. You can process identical elements in arrays by using either index manipulation or subscripting.

Bounds Checking

CA-Easytrieve automatically checks that indexes and subscripts do not reference data outside the storage boundary of the field being referenced. If your index or subscript is “out of bounds,” an execution error occurs. Subscripts are checked to ensure that they are within the OCCURS value of the field’s definition. Indexes are checked to ensure that the reference is within the largest enclosing data structure. For file fields, this structure is the file buffer. For working storage fields, this is the defined field, or the base field if the defined field is a redefinition.

On the workstation, bounds checking can have an impact on the performance of your program. See Coding an Efficient CA-Easytrieve Program later in this chapter for information on coding for optimum performance.

Indexing

Any data field definition can contain the INDEX attribute. An index can be used to reference data fields that occur multiple times. If you do not use an index, you must either use subscripts or assign individual field names to multiple field occurrences.

The data field’s starting location is adjusted by the contents of its indexes to determine the desired field occurrence. The INDEX *index-name* value is set to:

*(desired occurrence number - 1) * (length of element)*

Single Dimension Arrays

The following one-dimensional array is typical of those found in most programs. Data definition is straightforward. The value of MONTH-INDEX controls access to the desired data occurrence, MONTH.

Statements:

```

DEFINE ARRAY-ELEMENT W 2 N
DEFINE MONTHS      W 120 A VALUE +
                   ' JANUARY  +
                   FEBRUARY  +
                   MARCH     +
                   APRIL     +
                   MAY        +
                   JUNE       +
                   JULY       +
                   AUGUST     +
                   SEPTEMBER  +
                   OCTOBER   +
                   NOVEMBER  +
                   DECEMBER  '
DEFINE MONTH       MONTHS 10 A +
                   OCCURS (12) INDEX (MONTH-INDEX)
JOB INPUT NULL NAME MYPROG
  ARRAY-ELEMENT = 11
  MONTH-INDEX = (ARRAY-ELEMENT - 1) * 10
  DISPLAY MONTH

```

STOP

Results:

NOVEMBER

Since MONTH is 10 bytes long, the following relationships are true:

ARRAY-ELEMENT is	MONTH-INDEX is	DATA OCCURRENCE is
1	0	JANUARY
2	10	FEBRUARY
3	20	MARCH
...
12	110	DECEMBER

Multiple Dimension Arrays

Multiple dimension arrays can be defined in two different ways:

- Define a single field with multiple indexes.
- Index a re-defining field, as well as the parent field.

The following table illustrates two arrays which are identical in size and usage, but are defined very differently

MONTH-INDEX-1	MONTH-INDE X-2	MONTH	ROW-INDEX-1	COL-INDEX-2	MONTH-CELL
0	0	JANUARY	0	0	JANUARY
0	10	FEBRUARY	0	10	FEBRUARY
0	20	MARCH	0	20	MARCH
30	0	APRIL	30	0	APRIL
30	10	MAY	30	10	MAY
30	20	JUNE	30	20	JUNE
60	0	JULY	60	0	JULY
60	10	AUGUST	60	10	AUGUST
60	20	SEPTEMBER	60	20	SEPTEMBER
90	0	OCTOBER	90	0	OCTOBER

90	10	NOVEMBER	90	10	NOVEMBER
90	20	DECEMBER	90	20	DECEMBER

In both cases, the sum of the indices determines which data occurrence is referenced. Both MONTH and MONTH-CELL are ten-character fields with two indexes. Both fields also occur twelve times. MONTH-INDEX-1 and ROW-INDEX, and MONTH-INDEX-2 and COL-INDEX are considered similar indexes.

You can define and access arrays of more than two dimensions by a simple extension of the following examples.

Defining a Field with Multiple Indexes

Statements:

```

DEFINE QUARTER-ROW W 2 N
DEFINE MONTH-COL W 2 N
DEFINE MONTHS W 120 A VALUE +
    ' JANUARY +
    FEBRUARY +
    MARCH +
    APRIL +
    MAY +
    JUNE +
    JULY +
    AUGUST +
    SEPTEMBER +
    OCTOBER +
    NOVEMBER +
    DECEMBER '
DEFINE MONTH MONTHS 10 A OCCURS (12) +
    INDEX (MONTH-INDEX-1, MONTH-INDEX-2)
JOB INPUT NULL NAME MYPROG
    QUARTER-ROW = 4
    MONTH-COL = 2
    MONTH-INDEX-1 = (QUARTER-ROW - 1) * 30
    MONTH-INDEX-2 = (MONTH-COL - 1) * 10
    DISPLAY MONTH
    STOP
    
```

JANUARY	FEBRUARY	MARCH	
APRIL	MAY	JUNE	<— Quarter-Row
JULY	AUGUST	SEPTEMBER	<— Quarter-Row
OCTOBER	NOVEMBER	DECEMBER	<— Quarter-Row

```

↑      ↑      ↑
M      M      M
O      O      O
N      N      N
T      T      T
H      H      H
-      -      -
C      C      C
O      O      O
L      L      L
    
```

Result:

NOVEMBER

Redefining a Field, Giving each Field Its Own Index

Statements:

```

DEFINE QUARTER-ROW W 2 N
DEFINE MONTH-COL W 2 N
DEFINE MONTHS W 120 A VALUE +
    ' JANUARY +
    FEBRUARY +
    MARCH +
    APRIL +
    MAY +
    JUNE +
    JULY +
    AUGUST +
    SEPTEMBER +
    OCTOBER +
    NOVEMBER +
    DECEMBER '
DEFINE MONTH MONTHS 10 A +
OCCURS (12)
DEFINE MONTH-ROW MONTH 30 A, +
OCCURS 4, INDEX (ROW-INDEX)
DEFINE MONTH-COLS MONTH-ROW 10 A, +
OCCURS 3, INDEX (COL-INDEX)
DEFINE MONTH-CELL MONTH-COLS 10 A
JOB INPUT NULL NAME MYPROG
QUARTER-ROW = 4
MONTH-COL = 2
ROW-INDEX = (QUARTER-ROW - 1) * 30
COL-INDEX = (MONTH-COL - 1) * 10
DISPLAY MONTH-CELL
STOP
    
```

JANUARY (month-cell)	FEBRUARY (month-cell)	MARCH (month-cell)	← Month-Row
APRIL	MAY	JUNE	← Month-Row
JULY	AUGUST	SEPTEMBER	← Month-Row
OCTOBER	NOVEMBER	DECEMBER	← Month-Row

```

↑           ↑           ↑
M           M           M
O           O           O
N           N           N
T           T           T
H           H           H
-           -           -
C           C           C
O           O           O
L           L           L
S           S           S
    
```

Results:

NOVEMBER

Subscripts

Subscripts are an alternate method available to select an individual element from an array. A subscript is an integer (or a field containing an integer) that represents the occurrence number of the element within the array to be referenced. CA-Easytrieve computes the index value for you.

You can use subscripts with a field name in the following manner:

```
[file-name:] field-name ( subscript ... )
```

The following restrictions apply to the use of subscripts:

- A subscript must be a field name or a literal. An arithmetic expression cannot be coded for a subscript.
- A subscript's value must be a positive integer, no greater than the value specified for the OCCURS parameter of the DEFINE statement for *field-name*.
- You cannot subscript a field name used as a subscript.
- An indexed field cannot be used as a subscript.

Subscripting a One-Dimensional Array

A one-dimensional array is defined just as it would be if indexing were to be used. Referring to the Single Dimension Array shown earlier, the following table illustrates the relationship between the array element and the corresponding array element value:

ELEMENT is	VALUE is
MONTH(1)	JANUARY
MONTH(2)	FEBRUARY
MONTH(3)	MARCH
...	...
MONTH(12)	DECEMBER

For this array the maximum value that can be specified for the occurrence number is 12.

Subscripting a Two-Dimensional Array

A two-dimensional array is somewhat more complicated. To define a two-dimensional array, you must define the length and number of occurrences of each dimension. The following illustrates this:

```
DATA          W          30 A  VALUE 'AA+
                                     BB+
                                     CC+
                                     ...
                                     00'
```

```
ROW           DATA      10 A  OCCURS 3
COLUMN        ROW        2 A,  OCCURS 5
ELEMENT       COLUMN     2 A
```

This illustration defines a two-dimensional array (ELEMENT) with three rows and five columns, each occurrence of which is an alphabetic field of two characters. The first dimension (ROW) is defined as having three occurrences. The second dimension (COLUMN) is defined as having five occurrences. The length of the first dimension (ROW) must be the length of the second dimension (COLUMN) times the number of occurrences of the second dimension (COLUMN).

The next table illustrates the relationship between the array element and the corresponding array element value:

ELEMENT is	VALUE is
ELEMENT(1,1)	AA
ELEMENT(1,2)	BB
ELEMENT(1,3)	CC
ELEMENT(1,4)	DD
ELEMENT(1,5)	EE
ELEMENT(2,1)	FF
...	...
ELEMENT(3,5)	OO

Subscripting a Three-Dimensional Array

A three-dimensional array is a simple extension of a two-dimensional array. To define a three-dimensional array, you define the length and number of occurrences of each dimension (as you did for a two-dimensional array). The only difference is that you add the definition of a third dimension (MONTH-LET). This third dimension permits you to easily select individual positions within a cell in the array.

The following illustrates the definition and use of a three-dimensional array:

Statements :

```

DEFINE QUARTER-ROW W 2 N
DEFINE MONTH-COL W 2 N
DEFINE MONTHS W 120 A VALUE +
    ' JANUARY +
    FEBRUARY +
    MARCH +
    APRIL +
    MAY +
    JUNE +
    JULY +
    AUGUST +
    SEPTEMBER +
    OCTOBER +
    NOVEMBER +
    DECEMBER '
DEFINE MONTH-ROW MONTHS 30 A, +
    OCCURS 4
DEFINE MONTH-COLS MONTH-ROW 10 A, +
    OCCURS 3
DEFINE MONTH-LET MONTH-COLS 1 A, +
    OCCURS 10
DEFINE MONTH-CELL MONTH-LET 1 A
JOB INPUT NULL NAME MYPROG
* THIS PROGRAM DISPLAYS THE 3RD
* LETTER OF THE MONTH IN THE 4TH
* ROW, 2ND COLUMN (THE V IN NOVEMBER)
DISPLAY MONTH-CELL (4, 2, 3)
STOP

```

Results :

V

Segmented Data

One of the most common data structures is segmented data. Each record contains a fixed portion of data and multiple occurrences of data segments. The actual number of occurrences are not known until execution time. In COBOL, these structures are known as variable-length table definitions and are defined with an “occurs depending on” clause.

Defining Segmented Data

The following illustrates the field definitions necessary to describe a personnel record with a fixed area and variable occurrences of dependent and salary history segments:

```

FILE MASTER SEQUENTIAL
*
* FIXED PORTION
*
EMP-ID          1  5  N
EMPNAME        6 20  A
NO-OF-DEPENDS 26  2  N
NO-OF-JOBS     28  2  N
*
* DEPENDENT SEGMENTS
*
DEPEND-INFO    30 26  A      OCCURS 20
DEPEND-NAME    30 20  A      INDEX DEPINDEX
DEPEND-BIRTH   50  6  N      INDEX DEPINDEX

```

```

*
* SALARY HISTORY SEGMENTS
*
SALARY-HISTORY  30 16 A      OCCURS 10
  SALARY-AMOUNT 30  8 N  2    INDEX SALINDEX
  SALARY-GRADE  38  2 N      INDEX SALINDEX
  SALARY-EFF-DATE 40  6 N      INDEX SALINDEX

```

Because the starting location for each variable occurring segment is not known, the first position after the fixed portion is used. Later, to access the data, the length of the preceding segment(s) is added to the index to determine the starting location of the next variable segment. The OCCURS parameter specifies the maximum number of occurrences for each variable portion.

Accessing Segmented Data

The next example illustrates the index manipulation statements necessary to access the data contained in the file:

```

FILE MASTER SEQUENTIAL
*
* FIXED PORTION
*
EMP-ID           1  5  N
EMPNAME          6 20  A
NO-OF-DEPENDS   26  2  N
NO-OF-JOBS      28  2  N
*
* DEPENDENT SEGMENTS
*
DEPEND-INFO      30 26  A      OCCURS 20
  DEPEND-NAME    30 20  A      INDEX DEPINDEX
  DEPEND-BIRTH   50  6  N      INDEX DEPINDEX
*
* SALARY HISTORY SEGMENTS
*
SALARY-HISTORY  30 16  A      OCCURS 10
  SALARY-AMOUNT  30  8  N  2    INDEX SALINDEX
  SALARY-GRADE   38  2  N      INDEX SALINDEX
  SALARY-EFF-DATE 40  6  N      INDEX SALINDEX
WORK-CTR        W   2   N
*
JOB INPUT MASTER NAME PERSONNEL-REPORTS
MOVE ZEROS TO DEPINDEX, WORK-CTR . * INITIALIZE DEPENDENT INDEX,CTR
DO WHILE WORK-CTR < NO-OF-DEPENDS. * PROCESS ALL DEPENDENT PORTIONS
  PRINT DEPEND-REPORT
  WORK-CTR = WORK-CTR + 1
  DEPINDEX = DEPINDEX + 26
END-DO
*
MOVE ZERO TO WORK-CTR . * REINITIALIZE CTR
SALINDEX = (NO-OF-DEPENDS * 26) . * START OF SALARY HISTORY IS THE
* . * END OF THE DEPENDENT PORTION
DO WHILE WORK-CTR < NO-OF-JOBS . * PROCESS ALL SALARY PORTIONS
  PRINT SALARY-REPORT
  WORK-CTR = WORK-CTR + 1
  SALINDEX = SALINDEX + 16
END-DO
*
REPORT DEPEND-REPORT LINESIZE 72 SPACE 1
  TITLE 'DEPENDENT REPORT'
  LINE EMP-ID EMPNAME DEPEND-NAME DEPEND-BIRTHDATE
*
REPORT SALARY-REPORT LINESIZE 72 SPACE 1
  TITLE 'SALARY REPORT'
  LINE EMP-ID EMPNAME SALARY-AMOUNT SALARY-GRADE SALARY-EFF-DATE

```

Data Strings

Evaluating strings of data is another common index process. The following illustrates a technique for taking names from the input record, reversing them, and then printing them. The results of this program are:

REVERSED-NAME	DATA-NAME
GLORIA WIMN	WIMN,GLORIA
NANCY BERG	BERG,NANCY
GEORGE CORNING	CORNING,GEORGE
MARY NAGLE	NAGLE,MARY

The program code is as follows:

```

FILE NAMES CARD
  DATA-NAME      1      20      A
    SCAN-NAME     DATA-NAME  1      A  INDEX SUB1
  REVERSED-NAME   W      20      A
    SCAN-REVERSED REVERSED-NAME 1      A  INDEX SUB2
COUNTER          W      2      P  0
SAVE-COUNT       W      2      P  0
JOB INPUT NAMES
*
* INITIALIZE REVERSED NAME, SUB1, SUB2, AND COUNTER FIELDS
*
MOVE SPACES TO REVERSED-NAME
MOVE ZEROS TO SUB1, SUB2, COUNTER
*
* FIND LENGTH OF LAST NAME
*
  DO WHILE SCAN-NAME NQ ' ,'
    COUNTER = COUNTER + 1
    SUB1 = SUB1 + 1
  END-DO
SAVE-COUNT = COUNTER          . *SAVE LENGTH OF LAST NAME
COUNTER = 0                  . *RESET COUNTER
SUB1 = SUB1 + 1              . *BUMP SUB1 PAST THE COMMA
*
* FIND FIRST NAME AND MOVE TO REVERSED NAME
*
  DO WHILE SCAN-NAME NQ ' ' +
    AND COUNTER LE 20 - SAVE-COUNT - 1
    SCAN-REVERSED = SCAN-NAME
    COUNTER = COUNTER + 1
    SUB2 = SUB2 + 1
    SUB1 = SUB1 + 1
  END-DO
COUNTER = 0                  . *RESET COUNTER
SUB1 = 0                    . *RESET TO BEGINNING OF LAST NAME
SUB2 = SUB2 + 1             . *BUMP SO SPACE IS BETWEEN FIRST AND
*                            *LAST NAMES
* MOVE LAST NAME TO REVERSED NAME FIELD
*
  DO WHILE COUNTER LQ SAVE-COUNT - 1
    SCAN-REVERSED = SCAN-NAME
    COUNTER = COUNTER + 1
    SUB1 = SUB1 + 1
    SUB2 = SUB2 + 1
  END-DO
PRINT NAMES-REPORT
REPORT NAMES-REPORT LINESIZE 78
  TITLE 1 'EXAMPLE OF HOW TO REVERSE NAMES'
  TITLE 2 'INPUT FIELD FORMAT IS:'
  TITLE 3 'LAST-NAME, FIRST-NAME'
  LINE REVERSED-NAME DATA-NAME
END

```

Interprogram Linkage

The facilities of CA-Easytrieve provide all of the functions necessary to perform standard input/output, data examination, and data manipulation.

The LINK and TRANSFER statements can be used to invoke other CA-Easytrieve programs. You can also invoke subprograms written in other programming languages through the CALL, LINK, and TRANSFER statements, and the EXIT parameter of the FILE statement.

- The FILE EXIT and CALL statements enable you to invoke subprograms written in other programming languages. All discussions of the CALL statement also apply to FILE EXITs. (FILE EXITs are CALLs that are controlled automatically by CA-Easytrieve.)
- The LINK statement allows you to transfer control from the current program (parent) to another program (child) and then return control to the parent program.
- The TRANSFER statement allows you to transfer execution to a target program without returning to the invoking program.

CALL Statement on the Mainframe

The CALL statement on the mainframe provides a means to invoke subprograms written in other programming languages. The following topics discuss the mainframe techniques used with the CALL statement:

- Program linking
- Storage management
- Linkage (register usage) conventions
- Parameter list
- Error condition handling.

Program Linking

Called subprograms can be statically or dynamically linked with the CA-Easytrieve object module. You must declare which type of linkage you want to use in your CA-Easytrieve program with either the DECLARE statement or the CALL parameter on the PARM statement. For example:

```
PARM CALL (STATIC)
```

Or:

```
DECLARE INTCALC PROGRAM DYNAMIC
```

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.

CICS

In CICS, all dynamic programs are loaded by executing the CICS LOAD command. The LOAD command dynamically places the program in storage and returns the program's entry point to CA-Easytrieve.

Each time the CALL statement is executed, CA-Easytrieve determines whether or not the program has been loaded. If the program has not been loaded, CA-Easytrieve executes a CICS LOAD command to load it. Once loaded, the program remains loaded until one of the following points is reached:

- The end of the first activity that references the program – If the current activity (the child activity) was invoked with an EXECUTE statement from another activity (the parent activity), and if both the child and parent activity reference the program, the program is not deleted until the parent activity terminates. The termination of the child activity does not cause the program to be deleted.
- CA-Easytrieve performs the next screen input/output operation – If, however, you specified COMMIT NOTERMINAL on the SCREEN statement or the calling CA-Easytrieve program is executing conversationally, CA-Easytrieve does not delete the program.

MVS

In MVS, all dynamic programs are loaded by invoking the LOAD function of the operating system. The LOAD function dynamically places the program in storage and returns the program's entry point to CA-Easytrieve.

CA-Easytrieve loads the program as part of the initialization of the first activity that references the program. If the current activity (the child activity) was invoked with an EXECUTE statement from another activity (the parent activity), and if both the child and parent activity reference the program, then CA-Easytrieve loads the program during the initialization of the parent activity. In addition, CA-Easytrieve does not delete the program until the termination of the parent activity. Neither the initialization nor the termination of the child activity has any effect on the program's status.

Storage Management

In VSE, the author of programs in other languages is responsible for managing required storage. If additional storage is needed, (for example, to LOAD another program), you cannot use DOS COMREG facilities. All storage must be:

- Within the originally loaded program

- Obtained via GETVIS
- Uniquely controlled within the STORMAX area.

Linkage (Register Usage) Conventions

When CA-Easytrieve invokes a subprogram written in another programming language, it adheres to standard IBM register management conventions. The called subprogram must honor these conventions:

Register	Usage
REGISTER 1	Address of the parameter list
REGISTER 13	Address of an 18 fullword register save area
REGISTER 14	Address of where to return to within CA-Easytrieve
REGISTER 15	Address of the entry point in the subprogram

The subprogram must save the CA-Easytrieve registers in the save area addressed by REGISTER 13 and must restore them prior to returning via REGISTER 14. The 18 fullword register save area provided by CA-Easytrieve must be maintained as illustrated in the following table:

Save Area	Usage
WORD 1	Reserved
WORD 2	Set by CA-Easytrieve to address of the save area for the internal routine prior to the one issuing the subprogram call
WORD 3	Set by the subprogram to address of the save area within the subprogram
WORD 4 thru WORD 18	Set by the subprogram to values contained in CA-Easytrieve REGISTERS 14 through 12 upon entry to the subprogram.

Assembler Subprogram Linkage

Assembler language subprograms present no linkage problems. The following shows the instructions necessary to successfully control assembler language subprogram linkage:

```

ASMPGM CSECT
STM 14,12,12(13) save registers 14 through 12
LR 11,15 set base register
USING ASMPGM,11 assign base register
LA 14,0(0,13) address of CA-Easytrieve save area
LA 13,MYSAVE address of subprogram save area
ST 13,8(0,14) chain forward

```

```

      ST 14,MYSAVE+4 chain backward
      LR 10,1      save parameter list address
      ...
      ...
RETURN  L   13,4(0,13) address of CA-Easytrieve save area
      LM 14,12,12(13) restore CA-Easytrieve registers
      MVI 8(13),X'FF' indicate unused save area
      SR 15,15      set zero return code
      BR 14         return to CA-Easytrieve
      ...
MYSAVE DC 18A(0)    18 fullword save area
      ...
      ...

```

COBOL Subprogram Linkage

Note: COBOL subprograms are supported only in environments in which IBM supports a COBOL program invocation by an Assembler program. CICS does not support such an invocation.

COBOL subprogram linkage is dependent upon the operating system (OS/MVS or DOS/VSE) and the COBOL parameters that were in effect when the COBOL subprogram was compiled. Refer to the *COBOL Programmer's Guide* for specific details on these parameters and linkage conventions.

The following shows typical COBOL instructions necessary to control subprogram linkage:

```

      ...
LINKAGE SECTION.
01 PARAMETER-1.
      ...
01 PARAMETER-2.
      ...
01 PARAMETER-N.
      ...
PROCEDURE DIVISION USING PARAMETER-1,
                          PARAMETER-2,
                          ...
                          PARAMETER-N.
      ...
      ...
GOBACK
      ...

```

In DOS/VSE, for the subroutine to act correctly as a called program, the linkage control module ILBDMNS0 must be assembled as:

```

// EXEC ASSEMBLY
ILBDMNS0 CSECT
      DC X'FF000000000000000000000000000000'
* Older versions of COBOL used an eight byte ILBDMNS0
* Check existing link maps to determine the length.
      END
/*

```

CA-Easytrieve uses the dynamic call feature of OS/MVS to execute COBOL subprograms.

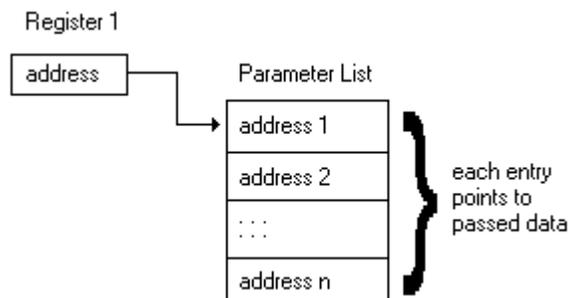
To establish a proper execution environment for COBOL subprograms in OS/VS COBOL, you can compile your COBOL subprogram with the NOENDJOB and NORESIDENT options. This establishes the subprogram properly as a called program. The NOENDJOB parameter is not available in COBOL II. Therefore, COBOL II subprograms are not supported in this version of CA-Easytrieve.

Note: You cannot CALL and LINK to an OS/VS COBOL program in the same activity.

Parameter Lists

The parameter list for both input/output and CALL exits (pointed to by register 1) passes information to the subprogram. Each entry in this contiguous group of fullwords identifies one parameter. The end of the list is indicated by the high-order bit of the high-order byte of the last entry being set to a one.

Parameter List Format



The parameter lists passed to subprograms for EXIT (FILE) and CALL are quite similar. In fact, the list for CALL is identical to that associated with the USING subparameter of EXIT. The only difference is that EXIT always passes at least two parameters.

Note: If multiple fields are coded on the USING subparameter, and storage areas overlap, results are unpredictable.

Error Condition Handling

Program errors which occur in subprogram exits cause the abnormal termination of CA-Easytrieve programs. Since these errors are occasionally difficult to analyze within the complex environment of CA-Easytrieve, exits should be tested first with simulation.

CALL Statement on the Workstation

The CALL statement on the workstation provides a means to invoke subprograms written in other programming languages. The following topics discuss the workstation techniques used with the CALL statement:

- Program linking
- Storage management
- Linkage conventions
- Error condition handling.

Program Linking

Called subprograms are statically linked with the CA-Easytrieve object module.

Storage Management

Called subprograms are free to use the INT 21H memory management services 48H (allocate) and 49H (free), the DosAllocSeq in MS-DOS or DosFreeSeq functions in OS/2, or the malloc() and free() 'C' functions.

Linkage Conventions

The CALL statement can be used to call programs that conform to either the FAR PASCAL or 'C' calling conventions. See the *CA-Easytrieve/Workstation User Guide* for more information.

FAR PASCAL Calling Convention

The FAR PASCAL calling convention is the calling convention normally used by PC-based ASSEMBLER, BASIC, FORTRAN, and PASCAL programs. The FAR PASCAL calling convention pushes the parameters onto the stack in the order in which they are coded on the USING clause. The called procedure must pop the parameters off of the stack before returning. This is the default calling convention.

'C' Calling Convention

The 'C' calling convention is used by workstation 'C' programs and most workstation COBOL programs. The 'C' calling convention pushes the parameters onto the stack in the reverse order in which they are coded on the USING clause. CA-Easytrieve pops the parameters off of the stack when the procedure returns.

Note: When compiling a CA-IDMS program, 'C' calling conventions are automatically used for all calls.

Stack Usage

For both the FAR PASCAL and 'C' calling conventions, the CALL statement pushes a far pointer (segment and offset) of each parameter coded onto the stack and expects that the called procedure is a far procedure. As such, the CALLED function or procedure must be compiled with a LARGE MEMORY MODEL specified.

The following diagram illustrates the contents of the stack for the CALL statement:

	----- STACK -----	
	FAR PASCAL	C
	-----	---
DEFINE FLDA W 5 A	CS	CS
DEFINE FLDB W 2 B	IP	IP
.	SEG FLDA	SEG FLDB
.	OFF FLDA	OFF FLDB
CALL TEST USING(FLDA FLDB)	SEG FLDB	SEG FLDA
	OFF FLDB	OFF FLDA

Note: 'C' compilers distributed by Borland usually require that an object module of startup code is incorporated into the executable. This can be accomplished at link-time. Refer to your Borland documentation for details.

If RETURNS is coded on the CALL statement, the value of the accumulator (AX) is placed into the RETURNS field. This is the standard return code convention.

Assembler Language Subprogram Example

Following is an example of a CA-Easytrieve program calling an Assembler subprogram:

```

DEFINE NUMBER W 2 I 0 VALUE 123
DEFINE RESULT W 2 I 0
JOB INPUT NULL
  DISPLAY 'NUMBER = ' NUMBER
  CALL SQUARE USING(NUMBER, RESULT)
  DISPLAY 'RESULT = ' RESULT
STOP

.SQUARE
.MODEL LARGE           ;large model.
.CODE                  ;code segment declaration.
PUBLIC SQUARE          ;make public for all.
PROC SQUARE            ;square procedure declaration.
  push bp               ;save base pointer.
  mov bp,sp             ;set stack pointer to base pointer.
  push ds               ;save CA-Easytrieve ds.
  mov ds,[bp+12]        ;get number to square segment.
  mov si,[bp+10]        ;get number to square offset.
  mov ax,[si]           ;get the number to square.
  mov cx,ax             ;save number to square in cx.
  mul cx                ;dx:ax = ax * cx.
  mov ds,[bp+8]         ;get result segment.
  mov si,[bp+6]         ;get result offset.
  mov [si],ax           ;move to CA-Easytrieve result field.
  pop ds                ;restore CA-Easytrieve ds.
  pop bp                ;restore original base pointer.
  ret 8                 ;pop 4 parms off the stack, return

```

```
SQUARE   ENDP           ;end of square procedure.
        END             ;end of module.
```

C Subprogram Linkage Example

Following is an example of a CA-Easytrieve program calling a 'C' function:

```

PARM CODE (PROCESS ASCII)
DEFINE FLDA W 2 I 0 VALUE 12345
DEFINE FLDB W 2 I 0 VALUE -23456
DEFINE FLDC W 2 I 0 VALUE 32767
DEFINE FLDD W 20 A VALUE('This is a test !')
DEFINE RC W 5 N
JOB INPUT NULL
  DISPLAY 'Ready to call C function FLDC =' FLDC
  CALL TEST1 USING(FLDA, FLDB, FLDC, FLDD) RETURNS(RC)
  DISPLAY 'Back from C function FLDC =' FLDC
  DISPLAY 'RETURN-CODE =' RETURN-CODE 'RC =' RC
  STOP

int TEST1(int *user1, int *user2, int *user3, char *field)
{
  char string[80];
  int i;
  printf("FLDA = %d\nFLDB =%d\nFLDC = %d\n", *user1, *user2, *user3);
  for(i=0; i<20; i++)
  {
    string[i]=*field;
    field++;
  }
  string[i]='\0';
  printf("FLDD = %s\n", string);
  *user3=9999;
  return(12345);
}

```

Exit Parameter List

You can use the EXIT parameter of the FILE statement to invoke subprograms written in other programming languages for input/output related events. Code the name of these subprograms on the EXIT parameter of the FILE statement in the library of your program.

Parameter List

address 1	}	-- address of work area
address 2		-- address of control code
...		-- optional USING parameters
address n		

For input/output exits, the work area address and the control code address are required parameters. The control code is a fullword used to indicate the function to be performed by the exit. For example:

Control Code Value	Function
00000000	input request

Control Code Value	Function
00000004	output request
00000008	file close request, or end-of-input (set by input exit subprogram)

For MODIFY exits (subparameter of the FILE statement), the required two parameters are record area address and work area address because the exit receives all records after input and before output.

Parameter List

address 1	} -- optional USING parameters	-- address of record
address 2		-- address of work area
...		
address n		

Parameters coded on the optional USING subparameter of EXIT are appended to the standard two parameters. The following exhibit shows input/output and MODIFY subprogram parameter list examples:

(INPUT/OUTPUT)

```
FILE USERFIL EXIT (ASMPGM USING (RECORD-LENGTH))
```

Parameter List

address of work area
address of control code
address of RECORD-LENGTH field

(MODIFY)

```
FILE USERFIL EXIT (ASMPGM MODIFY) WORKAREA(500)
```

Parameter List

address of record
address of work area

FILE EXIT Example

Following is an example of a program that calls a file exit written in PC Assembler:

```
FILE PERSNL F(150) EXIT MYEXIT
%PERSNL
JOB INPUT PERSNL
  DISPLAY REGION ' ' BRANCH ' ' EMPNAME
```

The file exit program code is as follows:

```

+-----+
| Procedure: MYEXIT.      Model: large.
|
| Sample file EXIT routine to get a record from the PERSNL file.
|
| parms passed = 4:
|
| [bp+12] - contains the segment of the file buffer.
|
| [bp+10] - contains the offset of the file buffer.
|
| [bp+8]  - contains the segment of the control code.
|
| [bp+6]  - contains the offset of the control code.
+-----+
MYEXIT_TEXT    segment          byte public 'code'
               assume          cs:MYEXIT_TEXT,ds:MYEXIT_DATA
MYEXIT         proc            far                ;exit routine.
               public          MYEXIT            ;make public for all.
               push            bp                ;save bp.
               mov             bp,sp             ;set bp to current sp.
               push            ax                ;save ax.
               push            bx                ;save bx.
               push            cx                ;save cx.
               push            dx                ;save dx.
               push            si                ;save si.
               push            di                ;save di.
               push            ds                ;save ds.
               push            es                ;save es.
               mov             ax,MYEXIT_DATA    ;MYEXIT data segment.
               mov             ds,ax             ;set ds to MYEXIT ds.
               cmp             OPEN_FLAG,0      ;file open ?
               jne            MYEXIT_010       ;yes, proceed.
               lea             dx,FILE_NAME     ;point to filename.
               mov             al,040h          ;R/O access deny none.
               mov             ah,03dh          ;open file function.
               int             21h              ;open the file.
               jc              MYEXIT_020       ;set EOF if error.
               mov             OPEN_FLAG,1      ;set file open flag.
               mov             FILE_HANDLE,ax   ;save file handle.
MYEXIT_010:    mov             bx,FILE_HANDLE   ;get file handle.
               mov             ds,[bp+12]       ;get buffer segment.
               mov             dx,[bp+10]       ;get buffer offset.
               mov             cx,150           ;read 150 bytes.
               mov             ah,03fh          ;read file function.
               int             21h              ;read the file.
               jc              MYEXIT_020       ;set EOF if error.
               cmp             ax,150           ;all bytes read ?
               je              MYEXIT_030       ;yes, end of case.
MYEXIT_020:    mov             ds,[bp+8]        ;control code segment.
               mov             di,[bp+6]        ;control code offset.
               mov             ax,00008h        ;close request code.
               mov             [di],ax          ;request EZT to close.
MYEXIT_030:    pop             es                ;restore es.
               pop             ds                ;restore ds.
               pop             di                ;restore di.
               pop             si                ;restore si.
               pop             dx                ;restore dx.
               pop             cx                ;restore cx.
               pop             bx                ;restore bx.
               pop             ax                ;restore ax.
               pop             bp                ;restore bp.
               ret             8                ;return to caller.
MYEXIT         endp                ;end of procedure.
MYEXIT_TEXT    ends
MYEXIT_DATA    segment
OPEN_FLAG      db              0                ;file open flag.
FILE_NAME      db              'PERSNL',0        ;filename for get.
FILE_HANDLE    dw              0                ;file handle.
MYEXIT_DATA    ends
end

```

Error Condition Handling

Program errors that occur in subprograms cause the abnormal termination of CA-Easytrieve programs. Because these errors can be difficult to analyze in the complex environment of CA-Easytrieve, subprograms should be tested before using them in CA-Easytrieve.

CALL Statement in UNIX

The CALL statement in UNIX provides a means to invoke subprograms written in other programming languages. The following topics discuss the UNIX techniques used with the CALL statement:

- Program linking
- Storage management
- Linkage conventions
- Error condition handling.

Program Linking

Called subprograms can be statically or dynamically linked with the CA-Easytrieve object module. You must declare which type of linkage you want to use in your CA-Easytrieve program with either the DECLARE statement or the CALL parameter on the PARM statement. For example:

```
PARM CALL (STATIC)
```

Or:

```
DECLARE INTCALC PROGRAM DYNAMIC
```

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.

Note: The AIX environment currently does not support DYNAMIC.

Storage Management

Called subprograms should use the malloc and free functions to allocate and free storage. The subprogram should free any storage it has allocated.

Linkage Conventions

The CALL statement can be used to call programs that conform to the 'C' calling conventions for the platform on which you are running CA-Easytrieve. See the *CA-Easytrieve for UNIX User Guide* for more information.

FILE EXIT Linkage

You can use the EXIT parameter of the FILE statement to invoke subprograms written in other programming languages for input/output related events. There are two types of exits you can code:

- Standard
- Using the MODIFY subparameter.

Standard Exits

A standard exit should perform its own I/O. A standard exit function should look like this:

```
unsigned long YourStandardExit(  
    long          eOperation,  
    void          * pRecord,  
    unsigned long * pcbRecord,  
    void          * pKey,  
    unsigned long * pcbKey,  
    P_EZEXIT_FCB pFCB,  
    void          * * pExitArgList  
);
```

where the meaning of parameters is as follows:

eOperation Has a value as shown in the following list of constants. These constants enumerate the possible values for *eOperation* and their meaning:

```
#define IO_OPEN          0 /* Open the file */  
#define IO_GET_NEXT_NH  4 /* GET NEXT NOHOLD */  
#define IO_GET_NEXT_H   6 /* GET NEXT HOLD */  
#define IO_GET_PRIOR_NH 10 /* GET PRIOR NOHOLD */  
#define IO_GET_PRIOR_H  12 /* GET PRIOR HOLD */  
#define IO_PUT          14 /* PUT */  
#define IO_READ_NH      18 /* READ NOHOLD */  
#define IO_READ_H       20 /* READ HOLD */  
#define IO_WRITE_ADD    22 /* WRITE ADD */  
#define IO_WRITE_UPD    24 /* WRITE UPDATE */  
#define IO_WRITE_DEL    26 /* WRITE DELETE */  
#define IO_POINT_FWD    28 /* POINT forward */  
#define IO_RELEASE      30 /* RELEASE */  
#define IO_CLOSE        42 /* CLOSE */  
#define IO_DISPLAY      48 /* DISPLAY */  
#define IO_SYNCPOINT    54 /* SYNCPOINT */  
#define IO_POINT_BWD    56 /* POINT backward */
```

pRecord Is the pointer to the record buffer. On input operations, your exit places data into this buffer from the file. On output, your exit writes the data from the buffer to the file.

pcbRecord Is the pointer to the length of the record currently in the buffer. On input operations, your exit must place the size of the record at this location.

pKey Is the pointer to the key. This field is valid only for operations that require a key.

pcbKey Is the pointer to the length of the key.

pFCB Is the pointer to the EXIT_FCB. The EXIT_FCB contains information describing the file. The EXIT_FCB is described after the description of the parameters.

pExitArgList Is the pointer to the array of pointers to the fields in the USING list of the EXIT phrase. The pointers are ordered as the fields in the USING list are ordered.

Return Value The following list of constants enumerate the valid return values and the meanings:

```
#define FC_NORMAL      0    /* The operation completed      */
/* normally.           */
#define FC_ENDFILE    4    /* The operation attempted to  */
/* position the file past the  */
/* last record in the file     */
#define FC_RDUPREC    8    /* The key of the record just   */
/* read matches the key of the  */
/* next record in the file     */
#define FC_WDUPREC   12    /* The operation attempted to  */
/* insert a record whose key   */
/* matches a record already in  */
/* the file                   */
#define FC_NRFOUND   16    /* The key specified for the   */
/* operation does not match    */
/* the key of any record in    */
/* the file                   */
#define FC_LOCKED    20    /* The operation attempted to  */
/* retrieve a record that is   */
/* locked by another user     */
#define FC_IOERROR   24    /* The operation failed for some */
/* reason other than one of   */
/* those given above         */
```

The ANCHOR is a control block that the exit might want to allocate. (It is not required.) Its purpose is to hold information that the exit needs from one invocation of the exit to the next. The ANCHOR is chained off of the EXIT_FCB (which follows).

Your exit is responsible for the creation, maintenance, and destruction of this area.

```
typedef struct ANCHOR {
    FILE          * pFile;
    /* More data could be placed here. Since there is no more data
     * in this example, a better solution would be to use the pAnchor
     * field in the EXIT_FCB as the file pointer.
    */
};
```

```
*/  
} ANCHOR, * P_ANCHOR;
```

The EXIT_FCB is a control block which is passed to the exit each time the exit is called. The same instance of the EXIT_FCB is passed from the time the file is opened to the time the file is closed for each operation on the file.

Hence, if your exit allocates its own control block (like the ANCHOR, shown below), its address can be placed as the first item in the EXIT_FCB and retrieved from the same place with each invocation of the exit. Remember to deallocate the control block when the file is closed.

CA-Easytrieve creates, maintains, and destroys this control block. Your exit should restrict itself solely to changing the *pAnchor* field.

```

typedef struct EZEXIT_FCB {
    P_ANCHOR      pAnchor;          /* for use by the Exit          */
    char          *pszFileName;     /* Pointer to the file name     */
    char          *pszPathName;     /* Pointer to the Path          */
    unsigned long cbRecordSize;     /* Logical record length        */
    unsigned long cbBufAreaLen;     /* Length of the buffer area    */
    void          *pBuffer;         /* Pointer to the file buffer   */
    char          szProcessMode[ 4 ]; /* File Access Mode             */
    unsigned char eRecordFormat;    /* Record format                */
#   define RF_NONE      0          /* Record format not specified */
#   define RF_FIXED     1          /* F or FB specified           */
#   define RF_VARIABLE  2          /* V, VB, or VSB specified     */
#   define RF_UNDEFINED 3          /* U specified                  */
    unsigned char eFileOrg;         /* File type                    */
#   define FO_NONE      0          /* File type not specified     */
#   define FO_SEQUENTIAL 1         /* SEQUENTIAL                  */
#   define FO_INDEXED  2          /* INDEXED                      */
#   define FO_RELATIVE 3          /* RELATIVE                    */
    unsigned char fMiscFlags0;     /* Miscellaneous flags          */
#if defined(LSB)
    /* For architectures with the Least Significant Byte First
     * such as the INTEL 80x86 chips.
     */
#   define FM_DEFER     0x01       /* DEFER                        */
#   define FM_ASA      0x02       /* ASA                          */
#   define FM_CREATE   0x04       /* CREATE                       */
#   define FM_RESET    0x08       /* CREATE RESET                 */
#   define FM_UPDATE   0x20       /* UPDATE                       */
#   define FM_NOVERIFY 0x40       /* NOVERIFY                     */
#   define FM_MODIFY   0x80       /* EXIT with MODIFY             */
#elif defined(MSB)
    /* For architectures with the Most Significant Byte First
     * such as the HP PA Risc (HP 9000) chips.
     */
#   define FM_DEFER     0x80       /* DEFER                        */
#   define FM_ASA      0x40       /* ASA                          */
#   define FM_CREATE   0x20       /* CREATE                       */
#   define FM_RESET    0x10       /* CREATE RESET                 */
#   define FM_UPDATE   0x04       /* UPDATE                       */
#   define FM_NOVERIFY 0x02       /* NOVERIFY                     */
#   define FM_MODIFY   0x01       /* EXIT with MODIFY             */
#endif
    unsigned char Reserved1;        /* Reserved for future expansion */
    void          *pReserved2;
    void          *pReserved3;
    void          *pReserved4;
} EZEXIT_FCB, * P_EZEXIT_FCB;

```

The Install Tape contains an example of a standard exit program, VRTXTEXT.c. Ask your system administrator where that program can be found.

MODIFY File Exit

If you code the MODIFY subparameter on the FILE statement, CA-Easytrieve performs the I/O. Your exit should examine the record and convert it into the correct form. On an input operation, CA-Easytrieve reads the record, then passes it to your MODIFY file exit to be reformatted. On an output operation, CA-Easytrieve passes the record to your exit to be reformatted. When your exit finishes, CA-Easytrieve writes the reformatted record to the file.

The prototype for your MODIFY file exit should look like:

```

unsigned long YRMDEXIT(
    void          * pRecordArea,
    void          * pWorkArea,
    unsigned long  cbRecordLength,

```

```
    unsigned long * pcbWorkAreaLength,  
    void * * pExitArgList  
);
```

where:

pRecord Points to the record buffer. The record to be reformatted is in this buffer.

pWorkArea Points to a buffer. The record from *pRecord* must be reformatted and placed in this buffer.

cbRecord This is the length of the record at *pRecord*.

pcbWorkArea Points to the length of *pWorkArea*'s buffer. Once your exit has placed the reformatted in the buffer, it must place the length of the reformatted record at this field.

pExitArgList Is the pointer to the array of pointers to the fields in the USING list of the EXIT phrase. The pointers are ordered as the fields in the USING list are ordered.

Return Value This exit uses the same values as were defined for the standard file exit.

The Install Tape contains an example of a MODIFY exit program, YRMDEXIT.c. Ask your system administrator where that program can be found.

If your exit processes variable length records, the RECORD-LENGTH field of the file should be in the USING list. Your exit must place the correct value in the USING list.

If your exit handles both input and output operations, you should place a field in the USING list that can tell your exit what type of operation the exit must perform.

Error Condition Handling

Program errors that occur in subprograms cause the abnormal termination of CA-Easytrieve programs. Because these errors can be difficult to analyze in the complex environment of CA-Easytrieve, subprograms should be tested before using them in CA-Easytrieve.

LINK Statement

The LINK statement is used to invoke another program and then return execution to the statement following the LINK statement in the original (invoking) program. The program invoked can be written in any language that is supported by the operating system in which the program is executing (including CA-Easytrieve).

LINK vs. CALL

The LINK statement differs from the CALL statement in the following ways:

- LINK creates a new program execution environment that bears a child relationship to the program issuing the LINK command (the parent program). A CALLED program executes in the same execution environment as the program issuing the CALL command.
- On the mainframe, a CALLED program uses the same linkage conventions in all execution environments, LINK uses the linkage conventions of the operating system in which the program is executing.

Commands Issued by the Child Program

The child program can issue any command supported by the operating system. In a CICS environment, the program can issue terminal I/O or display reports, but only in a fully-conversational mode.

Commands issued by a child program, such as SYNCPOINT, and I/O commands, can affect the operating environment of the parent program. CA-Easytrieve does not guarantee that applications using LINK execute identically in all execution environments. If portability between operating systems is required by an application, it is your responsibility to code the application in such a way that portability is assured.

USING Parameter

A single parameter can be passed to the child program by specifying the USING parameter. The parameter is passed to the child program by *value*, that is, the parent program passes a copy of the value of the field or literal to the child program. The child program cannot directly modify the value of the field or literal. You specify the field to receive the parameter with the USING parameter on the PROGRAM statement in the child program.

GIVING Parameter

You can allow the parent program to accept a return parameter from the child program by specifying the GIVING parameter. You specify the field to be returned in the GIVING parameter of the PROGRAM statement in the child program. CA-Easytrieve returns a value to the parent program by writing the return value into the same parameter area that was used by CA-Easytrieve to pass the USING parameter to the child program.

When the child program terminates, the parameter is copied from the parameter area to the variable specified in the GIVING parameter of the parent program. Because a single parameter area is used for communications in both directions, CA-Easytrieve determines the size of the parameter area by the larger of the fields specified on the USING and GIVING parameters. If the child program copies a return parameter into the parameter area with the PROGRAM GIVING parameter, but there is no GIVING parameter specified in the parent program, the returned parameter is ignored without any error indication.

Operating System Implementation

The LINK statement is implemented as listed below:

Operating System	Command	Parameters Passed By
CICS	EXEC CICS LINK	CICS communication area
TSO	MVS LINK SVC 6	Standard MVS/TSO linkage conventions
CMS	CMS LINK SVC 6	Standard CMS linkage conventions
PC/DOS	INTERRUPT 21H FUNCTION 4BH	PSP command tail or INT 60H
OS/2	DosExecPgm	Command line or shared segment
UNIX	system ("routine_name using_data")	

Note: When linking to an OS/VS COBOL program in TSO or CMS, the COBOL program must be compiled with the NORES compile option. Also, you cannot LINK and CALL an OS/VS COBOL program in the same activity.

Note: In CICS, the TRANSFER statement is more efficient than the LINK statement. In TSO and CMS, the opposite is true.

Workstation

The /FR (Far Reference) compiler switch is required to compile the LINKed program to CA-Easytrieve/Workstation if the PROGRAM statement has a USING field that is longer than 100 bytes or is a P, B, U, I, S, or D type field.

A secondary command processor (COMMAND.COM) is executed with the /C switch followed by the program name and any USING parameters. This method allows any .EXE, .COM, or .BAT file to be executed just as if the name were typed directly on the command line.

If the program name contains all blanks, a command shell is brought up on top of the parent process allowing the user to enter any valid DOS command. Type **EXIT** to terminate the shell and return control to the CA-Easytrieve parent program.

If the USING parameter is coded, the USING parameter is converted to ASCII, if necessary, and passed by value to the child process via the PSP command tail. If the /FR (Far Reference) compiler switch was specified, the USING data is passed to the child program by reference using the vector at Interrupt 60H, or via a shared segment in OS/2.

If the GIVING parameter is coded, the first item passed to the child process is the address of the GIVING field of the parent process. The CA-Easytrieve child program places the contents of the GIVING field specified on the program statement into the GIVING field of the parent process. The GIVING parameter should only be used with CA-Easytrieve child programs.

Note: The USING and GIVING parameters are limited to a maximum length of 100 characters, unless the child program was compiled with the /FR switch.

When the child process terminates with INT 21H function 4CH, or DosExit in OS/2, control is returned to the parent process.

Workstation Host Interface

CA-Easytrieve programs running on the workstation can communicate with a host TSO or CMS session using the LINK statement with the HOST parameter. The communication method uses the High Level Application Program Interface (HLLAPI). Requirements for this feature are:

- An IBM PC 3270 connection to the mainframe
- A PC 3270 emulator program that supports HLLAPI
- LLAPI executing resident on the workstation, or COMM MANNCU in OS/2.

See the *CA-Easytrieve/Workstation User Guide* for information on HLLAPI requirements and setup options.

The LINK statement sends the program name and USING parameter as an EBCDIC string to the host. Workstation execution continues immediately after the send operation is complete. The GIVING parameter is not returned to the workstation.

UNIX

The GIVING parameter on the LINK and PROGRAM statements is ignored in UNIX. The data from the USING parameter must be acceptable in the shell you are running.

TRANSFER Statement

The TRANSFER statement is used to transfer execution to a target program without returning to the invoking program. The target program can be written in any language that is supported by the operating system in which the program is executing.

The TRANSFER statement completely terminates the current CA-Easytrieve program and invokes a target program using the linkage conventions of the operating system in which the program is executing. The target program inherits the execution environment of the program issuing the TRANSFER command.

Commands Issued by the Target Program

The target program can issue any command supported by the operating system. In a CICS environment, the program can issue terminal I/O or display reports in a pseudo-conversational mode only if the program issuing the TRANSFER command can operate pseudo-conversationally.

Complete termination of the current CA-Easytrieve program does not imply the termination of the current logical unit of work. Although CA-Easytrieve attempts to terminate the current program (for example, close files, complete reports), it does not necessarily terminate all activities performed by the operating system for the CA-Easytrieve program.

Commands issued by a child program, such as SYNCPOINT, and I/O commands can affect the operating environment of the parent program. CA-Easytrieve does not guarantee that applications using TRANSFER execute identically in all execution environments. If portability between operating systems is required by an application, it is your responsibility to code the application in such a way that portability is assured.

USING Parameter

A single parameter can be passed to the target program by specifying the USING parameter. The parameter is passed to the target program by *value*, that is, the invoking program passes a copy of the value of the field or literal to the target program. The target program cannot directly modify the value of the field or literal. You specify the field to receive the parameter with the USING parameter on the PROGRAM statement in the target program.

Operating System Implementations

The TRANSFER statement is implemented as follows:

Operating System	Command	Parameters Passed By
CICS	EXEC CICS XCTL	CICS communication area
TSO	MVS XCTL SVC 7	Standard MVS/TSO linkage conventions
CMS	CMS XCTL SVC 7	Standard CMS linkage conventions
PC/DOS	INTERRUPT 21H FUNCTION 4BH	PSP command tail or INT 60H
OS/2	DosExecPgm	Command line or shared segment
UNIX	execl ("/bin/sh", "/bin/sh, "routine_name using_data", NULL)	

Note: When running pseudo-conversationally in CICS, you must specify the TRANSID parameter on the PARM statement in the target program. TRANSID specifies the CICS transaction ID that is invoked when the terminal user presses an attention key following the program's termination for a pseudo-conversation. The transaction ID of the target program must be defined in the PCT.

Note: In CICS, the TRANSFER statement is more efficient than the LINK statement. In TSO, CICS, and CMS, the opposite is true.

Workstation

The /FR (Far Reference) compiler switch is required to compile the LINKed program to CA-Easytrieve/Workstation if the PROGRAM statement has a USING field that is longer than 100 bytes or is a P, B, U, I, S, or D type field.

A secondary command processor (COMMAND.COM in PC/DOS, COMMAND.EXE in OS/2) is executed with the /C switch followed by the program name, the parent transfer buffer pointer and any USING parameters. This method enables any .EXE, .COM, or .BAT file to be executed just as if the name were typed directly on the command line.

If the USING parameter is coded, the USING parameter is converted to ASCII, if necessary, and passed by value to the child process using the PSP command tail in PC/DOS, or the command line in OS/2. If the /FR (Far Reference) compiler switch was specified, the USING data is passed to the child program by reference using the vector at Interrupt 60H in PC/DOS, or shared segment in OS/2.

Note: The USING and GIVING parameters are limited to a maximum length of 100 characters, unless the child program was compiled with the /FR switch.

If a TRANSFER command is issued by the child process, the TRANSFER command is placed into the transfer buffer of the parent process and the child execution is terminated with INT 21H function 4CH in PC/DOS, or DosExit in OS/2.

When the child process terminates with INT 21H function 4CH in PC/DOS, or DosExit in OS/2, control is returned to the parent process. If the parent process transfer buffer is empty, the parent process terminates via INT 21H function 4CH in PC/DOS, or DosExit in OS/2. If the parent process transfer command buffer is not empty, the parent process executes the command in the transfer buffer.

UNIX

If you coded a SHELL environment variable, CA-Easytrieve uses that path instead of "/bin/sh". The data from the USING parameter must be acceptable in the shell you are running.

CA-Easytrieve/ESP Interactive Execution

When a TRANSFER command is executed in CA-Easytrieve/ESP, the interpretive execution session is terminated.

Coding Efficient CA-Easytrieve Programs

Provided below are coding tips to help you write more efficient CA-Easytrieve programs.

Data Usage

- CA-Easytrieve normally generates the most efficient code when performing operations on binary fields (data type B) on the mainframe and integer fields (data type I) on the workstation. CA-Easytrieve performs binary or integer arithmetic whenever possible. When this is impractical due to the size of intermediate results, CA-Easytrieve uses packed decimal.
- When you code packed decimal fields, if you use 15 or fewer digits, CA-Easytrieve generates inline program code. On the mainframe, if you use more than 15 digits, CA-Easytrieve uses runtime library routines for multiplication and division. The use of runtime library routines results in a longer execution time of your program.
- Avoid the use of zoned numeric or fixed-point ASCII fields for heavily used computations in your program.
- Use indexes, rather than subscripts, in array processing to produce more efficient code. Although subscripts are easier to use, they must be internally computed into the index displacements.
- When you use subscripts to access arrays, use a two-byte binary field on the mainframe or an integer field on the workstation instead of a zoned numeric field to reduce execution time.
- CA-Easytrieve must convert one of the operands when you perform an operation on fields of different data types. To limit data conversions, code the fields to be updated or compared as the same type and the same number of decimal places whenever possible.
- When executing CA-Easytrieve on a workstation, limit the conversions that CA-Easytrieve must perform between EBCDIC and ASCII fields.
- In UNIX, all numeric operations require that the operands be converted to packed. Therefore, in the UNIX environment, packed decimal is the most efficient data type. However, this is subject to change.

Table Processing

- Avoid using very large tables because they take more time to search and require more storage than small tables.

Note: CA-Easytrieve automatically converts a SEARCH of an INDEXED table file into a keyed read when the ARG field is also the file's key. This results in much faster access and reduced storage requirements.

Compiler Site and Program Options

- The FASTSRT mainframe site option allows you to specify whether CA-Easytrieve instructs the mainframe sort program to process all of the I/O whenever possible. If the FASTSRT option is specified at your site, ensure that the sort program can process extended parameter lists.

- The STATE site option and PARM statement parameter saves the statement number of the statement currently being executed for display during an abnormal termination. This option/parameter requires approximately six to eight bytes of additional storage for each executable source line in your program.
- The FLOW site option and PARM statement parameter activates a trace of statements being executed to be printed during an abnormal termination. This option/parameter requires a subroutine to be invoked once for each executable source line.

After you test your program, deactivate the FLOW trace, if possible, to decrease execution time of your program.

- The FLDCHK site option and PARM statement parameter validates all data references during program execution. This option/parameter generates additional CA-Easytrieve code.

After you test your program, deactivate the FLDCHK validation, if possible, to decrease execution time of your program.

- The VFM site option and PARM statement parameter specifies the amount of storage available for the Virtual File Manager (VFM) buffer pool. Additional storage can decrease execution time.
- The BOUNCHK workstation option enables you to turn off bounds checking for subscripted and indexed field references. You can also turn off bounds checking with the /NB option during compilation.

Bounds checking generates a significant amount of additional code and subroutine usage on the workstation. To decrease execution time and storage consumption, deactivate bounds checking, if possible, after you test your program.

Report Processing

In non-CICS mainframe environments, report processing performance can be enhanced by coding the FILE parameter on the REPORT statement or the WORKFILE parameter of the PARM statement. Either of these parameters specifies that a dedicated work file is used rather than a VFM work file.

Coding Programs That Run Under CICS

Provided below are tips to help you write efficient CA-Easytrieve programs in a CICS environment.

- The following site options or PARM statement parameters are ignored under CICS:

- FASTSRT site option
- SSID site option
- SSID PARM statement parameter.
- CA-Easytrieve saves and restores all working storage fields, record I/O buffers, and table file entries across pseudo-conversations using temporary storage. Limit the number of fields and table entries you use in your program.
- To save on table usage, use INDEXED files as the subject of SEARCHes of large tables rather than instream files. When an INDEXED file is used as a table, the SEARCH statement results in a random read to the INDEXED file using the argument as the key.
- You must code the TRANSID parameter of the PARM statement if the program is invoked by another program through the TRANSFER statement. TRANSID specifies the PCT transaction ID that is invoked after a pseudo-conversation.
- Terminal I/O is done pseudo-conversationally unless COMMIT NOTERMINAL is specified on the activity statement. See Controlling Program Flow: Units of Work/Commit Processing earlier in this chapter for more information on CA-Easytrieve commit processing.

It is important to remember that commit points and pseudo-conversational terminal I/O can cause the following:

- VSAM file browses on non-unique paths are terminated.
- File holds are released.
- SQL cursors are closed and data is committed.
- CALLED subroutines are deleted and reloaded.
- CA-IDMS run units are terminated.
- DLI PSBs are terminated.

You must code your programs carefully. For example, when browsing an SQL file, you must reposition browses after a pseudo-conversation.

- Printer spool files are closed during each commit point, including pseudo-conversations.

Overview

CA-Easytrieve provides all the facilities necessary to process your file or database. Capabilities range from simple automatic input processing to complex controlled database maintenance. CA-Easytrieve processes the following file types:

- Sequential
- Indexed
- Relative record
- SQL, CA-IDMS, and IMS/DL/I databases
- BTRIEVE
- xBASE
- Lotus
- CA-SuperCalc
- Comma-delimited
- C-ISAM.

In addition, CA-Easytrieve provides its own sequential access method called the **Virtual File Manager** (VFM). VFM processes all the work files needed by a program.

This chapter describes the processing of sequential, indexed, and relative record files. It also includes a discussion of printer files, and a discussion of workstation file processing. See the “SQL Database Processing” chapter for a description of SQL processing. See the “CA-IDMS Database Processing” chapter for a description of CA-IDMS processing. See the “IMS/DL/I Database Processing” chapter for a description of IMS/DL/I processing.

Following are items that apply to each type of file that CA-Easytrieve processes.

File Definition

The FILE statement describes your file. It provides a logical name for the file and also the physical name of the file as known by the operating system. It can also specify the file's organization and characteristics. Following is an example of a FILE statement:

```

      (1)          (2)          (3)   (4) (5)  (6)          (7)
FILE PERSONNEL-MASTER SYSNAME 'PERSNL' SEQUENTIAL FB(150 1800) SYSTEM (PC PATH +
      'D:\DATA\PERSNL')
...
GET PERSONNEL-MASTER
      (1)

```

Refer to the numbers in the above exhibit:

- (1) The name of the file as it is known to the program.
- (2) The name that identifies the file to the operating system. SYSNAME is optional, and, if not specified, defaults to the file name (1). The name is one of the following depending on your environment:
 - A CMS FILEDEF name or DLBL name
 - An MVS DDname
 - A CICS FCT name
 - A fully-qualified workstation file name. For absolute portability when a workstation path is required, code the mainframe name here and also use the SYSTEM parameter (see (7) below).
 - A file description string or environment variable to the file (UNIX).
- (3) The organization type of the file: SEQUENTIAL, INDEXED, or RELATIVE.

Note: CA-Easytrieve Plus batch versions do not support the SEQUENTIAL, INDEXED, and RELATIVE keywords. For compatibility, these implementations support older FILE statements (VS or unspecified file types). See Appendix C in the *CA-Easytrieve Language Reference Guide* for differences.
- (4) The record format.
- (5) The record length.
- (6) The block size of the file. This is ignored on the workstation.
- (7) Environment-specific information can be specified in the SYSTEM parameter. This parameter is used to isolate information needed on the workstation. It is ignored on the mainframe and UNIX, enabling this FILE statement to be portable. See Workstation Files later in this chapter for more information on working with files on the workstation.

Note: CA-Easytrieve Plus batch versions 6.0 and below do not ignore the SYSNAME parameter and flag it as an error.

Controlled vs. Automatic Processing

File input can be either automatic or controlled.

JOB and SORT statements designate automatic input. SORT and the SUMFILE parameter of the REPORT statement are the only automatic output functions. All other output is, at least to some degree, under programmer control.

Input and output statements (DISPLAY, GET, CLOSE, POINT, PUT, READ, and WRITE) designate controlled processing. You can code these statements in any CA-Easytrieve processing activity, except SORTs, with or without automatic input.

Following are the rules for using automatic and controlled file processing:

- Controlled statements are not permitted in SORT or REPORT procedures.
- The GET statement cannot reference an automatic input file within the same JOB activity.
- Controlled statements are not normally valid for automatic input files in the JOB activity, with the following exceptions:
 - The POINT statement can be used in conjunction with automatic input for indexed and relative files. This allows for skip-sequential input processing while under system control.
 - The PUT and WRITE statements can be used for automatic input of an indexed or relative file, except when using synchronized file processing.

Data Buffering Mode

CA-Easytrieve always uses the move mode when processing files.

CA-Easytrieve moves the logical record from the buffer into a work area. File buffers and work areas are never initialized by CA-Easytrieve. Therefore, if you do not explicitly move data to the buffer or work area before output, the record can contain unpredictable data.

WORKAREA Parameter

Normally, CA-Easytrieve allocates the data buffer of a file either during the initiation of the activity that opens the file, or when the file is actually opened. If you reference the fields in a file before the buffer is allocated, a runtime error is issued indicating you have referenced a field in an unavailable file.

You can use the `WORKAREA` parameter on the `FILE` statement to allocate the buffer during the initiation of the program. The fields in the file will then always be available for reference during the execution of the program. These fields are treated similarly to working storage fields, however, unlike working storage fields, file fields are never initialized. If you reference these fields before a successful input operation, you must initialize them.

Record Format

File records must be in one of three formats:

- Fixed-length
- Variable-length
- Undefined-length.

Fixed-length and variable-length records can be blocked. Block sizes are ignored for Virtual File Manager (VFM) files and all files on the workstation.

All file formats must adhere to the standards established for processing by IBM input/output control system routines. Records that deviate from the standards for fixed-length or variable-length records can be processed only as undefined-length records.

See Workstation Files later in this chapter, for more information on record formats for workstation files.

See UNIX Files later in this chapter, for more information on record formats for UNIX files.

CARD, PUNCH, and VSAM

CA-Easytrieve makes the following assumptions about the format of CARD, PUNCH, and VSAM files:

- Mainframe CARD and PUNCH file records are always a fixed-length of 80 characters.
- Mainframe VSAM records have a variable length but no record length indicator.
- UNIX CARD files receive input from `stdin`.

When CA-Easytrieve produces variable-length or undefined-length records, the length of the output record is controlled by the current contents of the output file's `RECORD-LENGTH` field. Unless otherwise specified, all records created by CA-Easytrieve have a maximum data length based on the file's record-size attributes.

Record Address

The address of a record points to the first data byte of the record. The record-descriptor-word (RDW) of variable-length records is accessible only through the system-defined RECORD-LENGTH field.

STATUS Parameter

The STATUS parameter can be optionally specified on the GET, POINT, READ, PUT, and WRITE statements for files on which SEQUENTIAL, INDEXED, or RELATIVE is specified on the FILE statement.

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. You can code your program to perform an appropriate action based on that status.

STATUS causes the file's FILE-STATUS field to be set with the appropriate return code. See System-Defined File Fields below to determine the meaning of the contents of FILE-STATUS. Normally, a zero or non-zero test is sufficient. FILE-STATUS is not defined if you do not specify the file type on the file definition.

If you do not code STATUS and the operating system returns a non-zero status, CA-Easytrieve issues an appropriate diagnostic message.

System-Defined 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 four-byte binary 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.

RECORD-COUNT

RECORD-COUNT is a read-only four-byte binary field that contains the number of logical input operations performed to the file.

FILE-STATUS

FILE-STATUS is a read-only, four-byte binary field that contains a code that tells you the results of the most recent I/O operation on a file. See Appendix A, "System-Defined Fields," in the *CA-Easytrieve Language Reference Guide* for FILE-STATUS codes and their meanings.

Error Conditions

Error conditions that are flagged during file processing activities generally fall into one of the following three categories:

- **File OPEN errors** - commonly caused by incorrect or missing JCL or information (mainframe) or incorrect path information (workstation and UNIX). The operating system detects these errors and terminates CA-Easytrieve processing.
- **Invalid file reference errors** - caused by statements that refer to data from a file which does not have a current record, for example, after end-of-file or record not found. CA-Easytrieve issues a diagnostic message for these errors when the FLDCHK option is in effect.
- **Improper handling of nonzero STATUS conditions** - (returned from statements such as READ.) You are responsible for correctly resolving these conditions. (See System-Defined File Fields above for information on FILE-STATUS.)

Data Availability Tests

CA-Easytrieve provides conditional expressions to assist you in resolving questions of data availability. You can test these conditions with an IF statement after GET, POINT, READ, and WRITE statements and in association with synchronized file processing. The conditions used with synchronized file processing are described later in this chapter under *Synchronized File Processing*. The file presence test is available to test the success of the last input operation. For example, the IF *file-name* test is true when the last GET or READ operation was successful and there is another record that can be accessed.

Note: Referencing data in a file after any output operation is undefined and results are unpredictable unless you code WORKAREA on the FILE statement.

Opening Files

During the initiation of an activity, CA-Easytrieve opens all files used in the activity (except for those specified with the DEFER parameter) that are not already open. DEFERred files are opened when the first input/output statement is issued for them. Files that are already open when an activity begins remain open with the same file characteristics.

As part of its file-opening process, CA-Easytrieve performs the following tasks as needed:

- Validates block length for disk devices
- Sets the FULLTRK value
- Allocates buffer areas
- Allocates work areas
- Sets the file's processing and access modes.

Closing Files

CA-Easytrieve automatically closes all files that were opened by the activity at the end of each activity. You can also manually close a file by executing a CLOSE statement for the file. The next I/O statement using the file automatically reopens the file. Automatic input and output files cannot be manually closed.

Note: If you close an output file without executing a PUT statement, null files are created, unless you code DEFER on the FILE statement. Closing a DEFERred file leaves the file undefined.

File Processing Modes

CA-Easytrieve makes certain assumptions about the use of each file in an activity by scanning the I/O statements coded in the activity. (This is assuming that all statement syntax is valid.) One of these assumptions is the file's processing mode. The processing mode dictates whether the file is opened for read or write access. CA-Easytrieve file processing modes are:

- **Input** - indicates that only GET, POINT, and READ statements are coded in the activity that opened the file.
- **Create** - indicates that the file has CREATE specified on the FILE statement and that a PUT statement is coded in the activity. If you coded RESET on the FILE statement, CA-Easytrieve attempts to reload an existing file.
- **Output** - indicates that UPDATE is specified on the FILE statement and PUT or WRITE ADD statements are coded in the activity.

- **Update** - indicates that UPDATE is specified on the FILE statement and WRITE UPDATE or WRITE DELETE statements are coded in the activity that opened the file.

File Access Mode

The other assumption CA-Easytrieve makes about a file is the file's access mode. CA-Easytrieve always opens a file for dynamic access. This allows records to be read either sequentially or directly as follows:

- **Sequential** - indicates that records are accessed in a sequence determined by the file type as follows:
 - For file type SEQUENTIAL, the records are accessed in the sequence in which the records were added to the file.
 - For file type INDEXED, the records are accessed in the sequence defined by the key of the associated data set.
 - For file type RELATIVE, the records are accessed in ascending sequence of relative record numbers.

Sequential access allows the use of the GET, PUT, and WRITE UPDATE statements. If the file type is INDEXED or RELATIVE, sequential access also allows the use of the WRITE DELETE statement.

- **Direct** - indicates that records are accessed in a sequence determined by the program. Each statement that accesses a record specifies the key of the record to be accessed as follows:
 - For file type SEQUENTIAL, direct access is not allowed.
 - For file type INDEXED, the key value must be an alphanumeric item whose length is equal to the key length specified for the associated data set.
 - For file type RELATIVE, the key value is a 4-byte binary integer that specifies a relative record number.

Direct access allows the use of the READ, WRITE ADD, WRITE DELETE, and WRITE UPDATE statements.

Valid Syntax - FILE Statement

If file type is...	And parameters are...	I/O statements allowed:
SEQUENTIAL	CREATE UPDATE (none)	PUT GET, WRITE UPDATE GET

If file type is...	And parameters are...	I/O statements allowed:
RELATIVE	CREATE UPDATE (none)	PUT All except WRITE ADD GET, POINT, or READ
INDEXED	CREATE UPDATE (none)	PUT All GET, POINT, or READ
(none)	CREATE or UPDATE (none)	*** Error condition *** File type must be specified PUT or GET

File Browse Mode

Sequential input of a file implies a browse mode. CA-Easytrieve normally maintains the positioning within a file during a browse. The following operations terminate the browse and lose positioning within the file:

- A RELEASE statement
- Commit processing, automatic or by a COMMIT statement during a browse of a file containing a path of non-unique keys. See the “Units of Work/Commit Processing” chapter for more information.

Note: You can use the PRIOR parameter on the GET statement to perform a backward browse.

Hold/Release Processing

CA-Easytrieve automatically issues a hold for a record when the FILE statement specifies UPDATE. You can override this by specifying HOLD or NOHOLD on your READ and GET statements.

When CA-Easytrieve holds a record for update, it establishes the intent to update a record with the operating system. This intent does not mean you are obligated to actually perform the update. It just holds the position in the file and may also lock the record (CICS and workstation LANs). Locks are automatically released when the update operation completes or a commit point is taken.

You can manually release the hold on any file with the RELEASE statement.

In CICS, the default action for a READ statement is HOLD when you specify UPDATE on the FILE statement. The default action for a GET statement and for automatic input is NOHOLD. CICS does not allow you to browse a file for update.

In all other environments, CA-Easytrieve issues a HOLD by default for READs, GETs, and for automatic input. See Commit Processing in the “Screen Processing” chapter for examples of hold processing in screen processing.

Workstation LANs

CA-Easytrieve/Workstation supplies a mechanism to hold records in INDEXED files for update on LANs to protect records from concurrent update from multiple users. This facility functions similarly to the mainframe environment facility. When CA-Easytrieve issues a HOLD as described above, the record is locked from other users accessing the record. System options control how long subsequent users wait before being given an option to retry or escape the hold request. When the update is complete or when a commit point is taken, the record is automatically released.

SEQUENTIAL Files

CA-Easytrieve supports the access of all sequential files that the operating environment in which CA-Easytrieve is running also supports. When also supported by the operating environment, CA-Easytrieve supports all of the following sequential access methods:

- Queued Sequential Access Method (QSAM).
- Virtual Storage Access Method Entry Sequenced Data Set (VSAM ESDS).
- CARD and PUNCH files (not valid in CICS).
- CA-Easytrieve’s Virtual File Manager.
- Workstation fixed and variable-length text files, DBASE, LOTUS, CA-SuperCalc, comma-delimited files, and mainframe EBCDIC files through HLLAPI.
- In UNIX, variable-length text files and fixed-length record files are supported.

SEQUENTIAL File Processing Rules

CA-Easytrieve processes SEQUENTIAL files according to the following rules:

- When you do not code a file type on the FILE statement, SEQUENTIAL is the default. CARD, PUNCH, and VIRTUAL files imply a SEQUENTIAL file type.

Note: When SEQUENTIAL is not specified for a sequential file, the FILE-STATUS field is not available for the file.

- You cannot process the same SEQUENTIAL file as both an input and an output file within the same activity.
- You can create SEQUENTIAL files in one activity and process them by subsequent activities.
- CA-Easytrieve permits only one CARD file in a program.

SEQUENTIAL Input

CA-Easytrieve provides both automatic and controlled processing of SEQUENTIAL files.

Automatic Processing

The following example shows SEQUENTIAL processing with automatic control:

```
FILE SEQFILE FB(150 1800)
%PERSONL
JOB INPUT SEQFILE NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65
LINE EMP# EMPNAME
```

Controlled Processing

The next example shows SEQUENTIAL processing with programmer control:

```
FILE SEQFILE FB(150 1800)
%PERSONL
JOB INPUT NULL NAME MYPROG
GET SEQFILE
IF EOF SEQFILE
STOP
END-IF
PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65
LINE EMP# EMPNAME
```

CARD Input

In TSO or CMS, you can process one of the input files through the system input stream (SYSIN) by defining the file with a device type of CARD. On the workstation, CARD files are 80-byte card images.

In UNIX, the device type of CARD is equal to stdin. The file is treated as a variable-length text file.

Note: Unless redirected, stdin reads input from the terminal device.

```
FILE CARDFILE CARD
FIELD 1 5 A
JOB INPUT CARDFILE NAME MYPROG
DISPLAY FIELD
```

SEQUENTIAL Output

You can create output files using controlled processing with the PUT statement.

Fixed-length File Creation

The following example shows fixed-length SEQUENTIAL file creation:

```
FILE INFILE
FIELD 1 5 A
FILE OUTFILE FB(100 500)
FIELD 1 5 A
JOB INPUT INFILE NAME MYPROG
PUT OUTFILE FROM INFILE
```

Variable-length File Creation

The next example shows variable-length SEQUENTIAL file creation:

```
FILE INFILE
FIELD 1 5 A
FILE OUTFILE VB(100 504)
FIELD 1 5 A
JOB INPUT INFILE NAME MYPROG
OUTFILE:RECORD-LENGTH = 5
PUT OUTFILE FROM INFILE
```

VSAM File Creation

The FILE statement and the PUT statement are used to create (load) VSAM ESDS files. You can reference a newly created file in subsequent activities by coding another FILE statement with a different file name, but with JCL pointing to the same physical file. The example below illustrates reloading a fixed-length ESDS file.

You can create INDEXED and RELATIVE files using a similar technique. The data set must be defined as reusable to use the RESET option on the FILE statement.

```
FILE ESDS SEQUENTIAL CREATE RESET
%PERSNL
FILE PERSNL FB(150 1800)
COPY ESDS
JOB INPUT PERSNL NAME MYPROG
```

```

PUT ESDS FROM PERSNL STATUS
IF ESDS:FILE-STATUS NE 0
DISPLAY 'LOAD ERROR STATUS= ' ESDS:FILE-STATUS
STOP
END-IF

```

Note: When using multiple files, you should qualify FILE-STATUS, as illustrated above (ESDS:FILE-STATUS).

PUNCH Files

Except for the PUNCH parameter, CA-Easytrieve treats PUNCH files the same as any other 80-byte SEQUENTIAL file. The next example illustrates PUNCH file output:

```

FILE INFILE
FIELD 1 5 A
FILE OUTFILE PUNCH
FIELD 1 5 A
JOB INPUT INFILE NAME MYPROG
PUT OUTFILE FROM INFILE

```

In MVS, the PUNCH parameter is not required, due to its device independence. Simply define an output SEQUENTIAL file as fixed-length 80 characters and assign it to the proper SYSOUT class via the DD card. If you code the PUNCH parameter, MVS routes the output to the DD, SYSPUNCH.

Virtual File Manager

The Virtual File Manager (VFM) is a sequential access method that processes work files needed by a program. Typically, when work files are needed by a program, separate disk areas must be reserved for each work file. VFM, however, maintains as much disk area in memory as possible. If the area in memory is exhausted, VFM writes the excess data to a single spill area. When you use VFM, you need only define one physical file.

As a virtual file is read back into the program, the space it occupied is released and the area can be immediately reused. You can, however, retain VFM files for subsequent CA-Easytrieve activities.

The use of VFM is identical to SEQUENTIAL processing with the following additions:

- VFM files without the RETAIN option are deleted once CA-Easytrieve has processed them as input files and closed them.
- VFM files with the RETAIN option are deleted at the end of the associated CA-Easytrieve execution.
- VFM files are automatically blocked. Any block size you supply is ignored.

The following example illustrates a typical use of the VFM access method:

```
FILE PERSNL FB(150 1800)
%PERSNL
FILE SORTFILE VIRTUAL F(150)
COPY PERSNL
SORT PERSNL TO SORTFILE USING PAY-NET NAME MYSORT
JOB INPUT SORTFILE NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1
LINE EMPNAME PAY-NET
```

Because the file SORTFILE is a virtual file:

- You do not have to define SORTFILE in JCL
- SORTFILE does not leave any files known to the operating system.

INDEXED Files

CA-Easytrieve supports both sequential and random (direct) processing of INDEXED files. INDEXED files use a key that is contained in each record. Sequential processing retrieves records sequenced by the key. When also supported by the operating environment, CA-Easytrieve supports the following index access methods:

- Virtual Storage Access Method Keyed Sequenced Data Set (VSAM KSDS)
- Workstation Fast Access Btree Structure (FABS) ISAM files
- Workstation BTRIEVE files (fixed format only)
- UNIX C-ISAM files (fixed format, unique single keys only).

INDEXED Sequential Input

CA-Easytrieve can process INDEXED files as if they were SEQUENTIAL files. The access can be done either automatically (JOB INPUT) or controlled (GET). The process used is essentially the same as that described for SEQUENTIAL file input. Additionally, you can also start the input at a specific record or use skip-sequential processing to bypass groups of records. The following two examples illustrate INDEXED sequential input with a starting record and skip-sequential processing.

POINTing to a Specific Record

```
FILE KSDS INDEXED F(150)
%PERSNL
JOB INPUT KSDS NAME MYPROG START POINT-PROC
DISPLAY EMP# +2 EMPNAME
POINT-PROC. PROC
POINT KSDS EQ '12318' STATUS
IF FILE-STATUS NE 0
```

```

        DISPLAY 'FILE-STATUS= ', KSDS:FILE-STATUS
        STOP
    END-IF
END-PROC

```

Skip-Sequential Processing

```

FILE VSAM INDEXED
%PERSNL
JOB INPUT VSAM NAME MYPROG
  IF EMP# EQ 1000 THRU 1999
    PERFORM POINT-VSAM
    GOTO JOB
  END-IF
PRINT REPORT1
*
POINT-VSAM. PROC
  POINT VSAM GE '02000' STATUS
  IF VSAM:FILE-STATUS NE 0
    DISPLAY 'FILE-STATUS= ' FILE-STATUS
  END-IF
END-PROC
*
REPORT REPORT1 LINESIZE 65
  LINE EMP# EMPNAME

```

Random Input

INDEXED files can be read randomly (direct access mode) by the READ statement. The next example illustrates reading an INDEXED file (PERSNL) using keys contained in a SEQUENTIAL file (INKEYS).

```

FILE PERSNL INDEXED
%PERSNL
FILE INKEYS
WHO * 5 N
JOB INPUT INKEYS NAME MYPROG
  READ PERSNL KEY WHO STATUS
  IF FILE-STATUS EQ 16
    DISPLAY 'BAD KEY=' WHO
  GOTO JOB
  END-IF
DISPLAY SKIP 2 HEX PERSNL

```

Adding Records to an INDEXED File

You can use either the WRITE or PUT statements to add records to an INDEXED file. Either statement adds a single record to the file, but to use mass-sequential insertion, use the PUT statement to add many records to the same place in the file.

If you use the WRITE or PUT statements, you must include the CREATE or UPDATE parameter on the FILE statement. UPDATE informs CA-Easytrieve that all input records can potentially be updated or deleted. CREATE informs CA-Easytrieve that the activity will use the PUT statement to load the file.

The following examples illustrate single and mass-insertion record addition.

Adding a Single Record

```

FILE PERSNL INDEXED UPDATE
%PERSNL
FILE INKEYS
WHO * 5 N
PHONE * 10 N
JOB INPUT INKEYS NAME MYPROG
  MOVE WHO TO EMP#
  MOVE PHONE TO TELEPHONE
  WRITE PERSNL ADD STATUS
  IF PERSNL:FILE-STATUS NE 0
    DISPLAY 'BAD KEY=' WHO ' STATUS=' PERSNL:FILE-STATUS
  GOTO JOB
END-IF

```

Mass-Sequential Record Insertion

```

FILE PERSNL INDEXED UPDATE
%PERSNL
FILE LOADER
COPY PERSNL
JOB INPUT LOADER NAME MYPROG
  PUT PERSNL FROM LOADER STATUS
  IF FILE-STATUS NE 0
    DISPLAY 'ADD FAILED'
    DISPLAY HEX LOADER
  STOP
END-IF

```

File Creation

The FILE statement and the PUT statement are used to create (load) INDEXED files. You can reference a newly created file in subsequent activities by coding another FILE statement with a different file name, but with JCL that points to the same physical file. The following example illustrates reloading a INDEXED file. The data set must be defined as reusable to use the RESET option on the FILE statement.

```

FILE KSDS INDEXED CREATE RESET
%PERSNL
FILE PERSNL FB(150 1800)
COPY ESDS
JOB INPUT PERSNL NAME MYPROG
  PUT KSDS FROM PERSNL STATUS
  IF KSDS:FILE-STATUS NE 0
    DISPLAY 'LOAD ERROR STATUS=' ESDS:FILE-STATUS
  STOP
END-IF

```

Deleting a Record

You can use the WRITE statement to delete individual records from an INDEXED file. The deleted record is the file's current input record.

```

FILE KSDS INDEXED UPDATE
%PERSNL
FILE KEYS
WHO 1 5 N
JOB INPUT KEYS NAME MYPROG
  READ KSDS KEY WHO STATUS
  IF FILE-STATUS NE 0

```

```

        DISPLAY 'READ FAILED...KEY= ' WHO
        STOP
    END-IF
WRITE KSDS DELETE STATUS
IF FILE-STATUS NE 0
    DISPLAY 'DELETE FAILED'
    STOP
END-IF

```

Updating a Record

You can modify and rewrite the current input record by using the WRITE statement.

```

FILE KSDS INDEXED UPDATE
%PERSONL
FILE KEYS
WHO      1  5 N
PHONE   6 10 N
JOB INPUT KEYS NAME MYPROG
READ KSDS KEY WHO STATUS
IF FILE-STATUS NE 0
    DISPLAY 'READ FAILED...KEY= ' WHO
    STOP
END-IF
MOVE PHONE TO TELEPHONE
WRITE KSDS UPDATE STATUS
IF FILE-STATUS NE 0
    DISPLAY 'UPDATE FAILED...KEY= ' WHO
    STOP
END-IF

```

RELATIVE Files

CA-Easytrieve supports both sequential and random (direct file access) processing of RELATIVE files. RELATIVE files use a four-byte binary key that contains an integer value that specifies the relative record number. Sequential processing retrieves records sequenced by the relative sequence number. When also supported by the operating environment, CA-Easytrieve supports the following relative access methods:

- Virtual Storage Access Method Relative Record Data Set (VSAM RRDS)
- Workstation and UNIX fixed-length relative (random-access) files.

RELATIVE File Sequential Input

CA-Easytrieve can process RELATIVE files as if they were SEQUENTIAL files. The access can be done either automatically (JOB INPUT) or controlled (GET). The process used is essentially the same as that described for SEQUENTIAL file input. Additionally, you can start the input at a specific record or use skip-sequential processing to bypass groups of records.

POINTing to a Specific Record

The following example shows RELATIVE sequential input with a starting record:

```
FILE RRDS RELATIVE
%PERSNL
REC-NUMBER W 4 B
JOB INPUT RRDS NAME MYPROG START POINT-PROC
  DISPLAY EMP# +2 NAME
POINT-PROC. PROC
  REC-NUMBER = 20
  POINT RRDS EQ REC-NUMBER STATUS
  IF FILE-STATUS NE 0
    DISPLAY 'FILE-STATUS= ', RRDS:FILE-STATUS
  STOP
END-IF
END-PROC
```

Skip-Sequential Processing

The next example shows RELATIVE skip-sequential processing:

```
FILE RRDS RELATIVE
%PERSNL
REC-NUMBER W 4 B
JOB INPUT RRDS NAME MYPROG
  IF RECORD-COUNT GR 3
    PERFORM POINT-RRDS
    GOTO JOB
  END-IF
  PRINT REPORT1
*
POINT-RRDS. PROC
  REC-NUMBER = 40
  POINT RRDS GE REC-NUMBER STATUS
  IF RRDS:FILE-STATUS NE 0
    DISPLAY 'FILE-STATUS= ' FILE-STATUS
  END-IF
END-PROC
*
REPORT REPORT1 LINESIZE 65
  LINE EMP# EMPNAME
```

Random Input

RELATIVE files can be read randomly (direct access mode) by the READ statement. The example below illustrates reading a RELATIVE file, PERSNL, using keys contained in a SEQUENTIAL file, INKEYS:

```
FILE PERSNL RELATIVE
%PERSNL
FILE INKEYS
WHO * 4 B
JOB INPUT INKEYS NAME MYPROG
  READ PERSNL KEY WHO STATUS
  IF FILE-STATUS NE 0
    DISPLAY 'BAD KEY=' WHO
  GOTO JOB
END-IF
DISPLAY SKIP 2 HEX PERSNL
```

Adding Records to a RELATIVE file

You can use the PUT statement to add records to a RELATIVE file. If you use the PUT statement, you must include the CREATE or UPDATE parameter on the FILE statement. UPDATE informs CA-Easytrieve that all input records can potentially be updated or deleted. CREATE informs CA-Easytrieve that the activity will use the PUT statement to load the file. The following examples illustrate record addition and file creation.

Record Addition

```
FILE PERSNL RELATIVE UPDATE
%PERSNL
FILE LOADER
COPY PERSNL
REC-NUMBER W 4 B VALUE 330. * STARTING SLOT NUMBER
JOB INPUT LOADER NAME MYPROG
  POINT PERSNL GE REC-NUMBER STATUS
  PUT PERSNL FROM LOADER STATUS
  IF FILE-STATUS NE 0
    DISPLAY 'ADD FAILED'
    DISPLAY HEX LOADER
  STOP
END-IF
REC-NUMBER = REC-NUMBER + 1
```

File Creation

The FILE statement and the PUT statement are used to create (load) RELATIVE files. You can reference a newly created file in subsequent activities by coding another FILE statement with a different file name, but with JCL that points to the same physical file. The example below illustrates reloading a RELATIVE file. The data set must be defined as reusable to use the RESET option on the FILE statement:

```
FILE RRDS RELATIVE CREATE RESET
%PERSNL
FILE PERSNL FB(150 1800)
COPY RRDS
JOB INPUT PERSNL NAME MYPROG
  PUT RRDS FROM PERSNL STATUS
  IF RRDS:FILE-STATUS NE 0
    DISPLAY 'LOAD ERROR STATUS= ' RRDS:FILE-STATUS
  STOP
END-IF
```

Deleting a Record

You can use the WRITE statement to delete individual records from a RELATIVE file. The deleted record is the file's current input record. For example:

```
FILE RRDS RELATIVE UPDATE
%PERSNL
FILE KEYS
WHO 1 4 B
JOB INPUT KEYS NAME MYPROG
  READ RRDS KEY WHO STATUS
  IF FILE-STATUS NE 0
```

```

        DISPLAY 'READ FAILED...KEY= ' WHO
        STOP
    END-IF
    WRITE RRDS DELETE STATUS
    IF FILE-STATUS NE 0
        DISPLAY 'DELETE FAILED'
        STOP
    END-IF

```

Updating a Record

You can modify and rewrite the current input record by using the WRITE statement. For example:

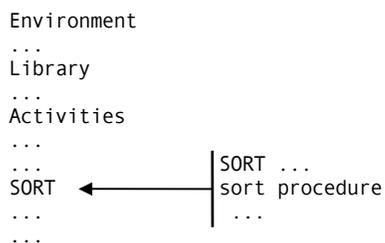
```

FILE RRDS RELATIVE UPDATE
%PERSONL
FILE KEYS
WHO      1  4  B
PHONE   6 10  N
JOB INPUT KEYS NAME MYPROG
  READ RRDS KEY WHO STATUS
  IF FILE-STATUS NE 0
    DISPLAY 'READ FAILED...KEY= ' WHO
    STOP
  END-IF
  MOVE PHONE TO TELEPHONE
  WRITE RRDS UPDATE STATUS
  IF FILE-STATUS NE 0
    DISPLAY 'UPDATE FAILED...KEY= ' WHO
    STOP
  END-IF

```

Sorting Files

You can use the SORT statement to sort files. CA-Easytrieve can sort any file that can be processed sequentially. The following illustrates the position of a SORT activity within a CA-Easytrieve program:



Your installation's sort program performs the actual sort process, except in CICS, on the workstation, and in UNIX, where CA-Easytrieve supplies its own sort program. CA-Easytrieve normally utilizes conventional sort interface techniques. For example, on a mainframe CA-Easytrieve invokes the sort program's E15 (input) and E35 (output) exits.

You can set a site option to use versions of sort in which the sort processes all input and output. For detailed information on the available options for sort program utilization, refer to your installations's sort program manual.

Note: In CICS, on the workstation, and in UNIX, CA-Easytrieve provides a sort program.

See Sequenced Reports in the "Report Processing" chapter for information on sorting report output.

SORT Activity 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
JOB INPUT SORTWRK NAME MYPROG
  PRINT REPORT1
*
REPORT REPORT1
LINE REGION BRANCH EMPNAME
```

Sort Procedures

CA-Easytrieve normally sorts all input records and outputs them to the file you specify. The output file usually has the same format and length as the input file. However, sometimes you may want to sort only certain records and/or modify the contents. To do this, you can write a sort procedure which must immediately follow the SORT statement.

You can code any valid CA-Easytrieve statement in a sort procedure, but you cannot code statements that generate input/output. Invalid statements are:

```
COMMIT      DELETE
DISPLAYFETCH  GET
IDMS  INSERT
POINT PRINT
PUT  READ
RETRIEVE (IDMS)  ROLLBACK
SELECT (SQL)  SELECT (IDMS)
SQL  UPDATE
WRITE
```

Note: For debugging purposes, you can DISPLAY to the system output device (SYSPRINT/SYSLST).

The only valid field references within a sort procedure are:

- Any field of the input file
- Any working storage field
- System-defined fields such as SYSDATE and RECORD-LENGTH.

Sorting a Selected Portion of a File

CA-Easytrieve supplies input records to your sort procedure one at a time. If you code a BEFORE procedure, a SELECT statement must be executed for each record that you want to sort.

The following example illustrates how to code an output file that will contain 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
*
JOB INPUT SORTWRK NAME MYPROG
  PRINT REPORT1
*
REPORT REPORT1
LINE REGION BRANCH DEPT NAME-LAST NAME-FIRST
```

Synchronized File Processing

The Synchronized File Processing (SFP) facility can be used with one file or multiple files:

- Synchronized File Input - performs match/merge operations on multiple files.
- Single File Keyed Processing - compares the contents of a key field or fields from one record to the next in a single file.

Synchronized File Input

CA-Easytrieve has a twofold solution to help you avoid coding complex logic for match/merge operations:

- Automatic input that includes a universally-adaptable match/merge algorithm.
- Special conditional expressions that help to determine simple, yet precise file relationships.

The synchronized file match/merge algorithm is based on the following assumptions and rules:

- Two or more files capable of being processed sequentially can be accessed.
- All files involved in the operation must be in ascending order by their key values.
- The number of keys for each file is identical.
- Corresponding keys of all files must be either alphanumeric or numeric. An alphanumeric key must be alphanumeric in all files, but can have different lengths. A numeric key must be numeric in all files, but can have different data types (N, P, U, B, F, I) and lengths.
- Because the algorithm must “read ahead” to perform a match/merge, INDEXED, RELATIVE, and input files cannot be updated during synchronized file processing.
- You can use the POINT statement to position an INDEXED or RELATIVE file at a record other than the first record before processing. Use a START procedure to perform the positioning.

Example

The INPUT parameter of the JOB statement designates files and their keys for synchronized file input. The example below illustrates a variety of synchronized file and key combinations:

```

FILE FILE1  ...
KEY1A  1  5  A
KEY1B  6  4  P
...
KEY1X  ...
...
FILE FILE2  ...
KEY2A  24 5  A
KEY2B  1  2  B
...
KEY2X  ...
...
FILE FILEN  ...
KEYNA  17 5  A
KEYNB  10 7  N
...
KEYNX  ...
...

```

```

JOB INPUT(FILE1 KEY KEY1A +
          FILE2 KEY KEY2A)
...
JOB INPUT (FILEN KEY(KEYNB, KEYNA) +
          FILE1 KEY(KEY1B, KEY1A) +
          FILE2 KEY(KEY2B, KEY2A))
...
JOB INPUT (FILE1 KEY(KEY1A ...) +
          ...
          FILEN KEY(KEYNA ...))
...

```

Record Availability

Records from files in CA-Easytrieve synchronized file input are made available for processing based on the relationship of the files' keys. Records with the lowest keys are made available first, and the match is hierarchical based upon the order of the files specified on the JOB statement.

Refer to the following three input files for an example of synchronized file input:

FILE1	FILE2	FILE3
1	2	1
2	3 A	3
3 A	3 B	4
3 B	4 A	5
8 A	4 B	7
8 B	6	8 A
9	7	8 B

The key is the single numeric digit and the letter indicates duplicates for illustrative purposes. The JOB statement to process the three files for the match/merge operation is:

```

JOB INPUT (FILE1 KEY(KEY1) +
          FILE2 KEY(KEY2) +
          FILE3 KEY(KEY3)) NAME MYPROG

```

Duplicate key values affect record availability differently based on which file contains the duplicates. Remember, the matching algorithm is hierarchical so the key is exhausted on the lowest level before another record is processed from the next higher level file.

Match/Merge Operation Output

The following exhibit illustrates the match/merge output from the synchronized file input process. The output shows the results of each iteration (loop) through the JOB activity. **N/A** under a file indicates that a record from the file is not available and no fields from this file can be referenced during the associated iteration.

Note: CA-Easytrieve provides special IF statements to help you determine record availability. See Special IF Statements next.

JOB ITERATION	FILE1 RECORD	FILE2 RECORD	FILE3 RECORD
1	1	N/A	1
2	2	2	N/A
3	3A	3A	3
4	3A	3B	N/A
5	3B	N/A	N/A
6	N/A	4A	4
7	N/A	4B	N/A
8	N/A	N/A	5
9	N/A	6	N/A
10	N/A	7	7
11	8A	N/A	8A
12	8A	N/A	8B
13	8B	N/A	N/A
14	9	N/A	N/A

Refer to iterations 3 through 5 in the above output. FILE1 and FILE2 both contain two records with a key value of 3. FILE3 contains only one record of key 3. These records are processed by CA-Easytrieve, as follows:

- **Iteration 3:** The first record 3 from FILE1 and FILE2 and the only record with key 3 from FILE3 are available.
- **Iteration 4:** Since the next record on FILE3 is a key 4 record and there are still key 3 records to process in the other files, FILE3's record is not available. CA-Easytrieve goes back to FILE2 and gets the next key 3 record. The original key 3 record from FILE1 and the second key 3 record from FILE2 are available.
- **Iteration 5:** Since the next record on FILE2 is a key 4 record and there is still a key 3 record on FILE1 to process, FILE2 is now unavailable. CA-Easytrieve returns to FILE1 and retrieves the next record. This time the only record available is the second key 3 record from FILE1.

Special IF Statements

CA-Easytrieve provides a simple way of determining the contents of current synchronized file input with special conditional expressions.

MATCHED

Use the MATCHED test to determine the relationship between the current record of one file and the current record of one or more other files.

```
IF [NOT] MATCHED [file-name-1 ... file-name-2 ...]
```

Refer to the earlier table under Match/Merge Operation Output, which depicts automatic synchronized file input.

- IF MATCHED is true for JOB iteration 3.
- IF MATCHED FILE1, FILE3 is true for JOB iterations 1, 3, 11, and 12.

- IF MATCHED FILE2, FILE3 is true for JOB iterations 3, 6, and 10.

File Existence

To determine the presence of data from a particular file, use the following special conditional expressions:

```
IF [NOT] file-name
IF [NOT] EOF file-name
```

When the IF *file-name* condition is true, a record from that file is present and available for processing. The IF NOT *file-name* condition is true when the file does not contain a record with a current key. When this condition is true, no fields from the file can be referenced in the activity. If you reference a field in an unavailable file, CA-Easytrieve issues a runtime error.

Refer again to the earlier table under Match/Merge Operation Output.

- IF FILE1 is true for JOB iterations 1 through 5 and 11 through 14.
- IF NOT FILE2 is true for JOB iterations 1, 5, 8, and 11 through 14.
- IF EOF FILE2 is true for JOB iterations 11 through 14.

DUPLICATE, FIRST-DUP, and LAST-DUP

The DUPLICATE, FIRST-DUP, and LAST-DUP tests determine the relationship of the current record of a file to the preceding and following records in the same file:

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

The following record relationship tests are based on the previous example of automatic synchronized file input. See the table under Match/Merge Operation Output.

- IF DUPLICATE FILE1 is true for JOB iterations 3 through 5 and 11 through 13.
- IF FIRST-DUP FILE2 is true for JOB iterations 3 and 6.
- IF LAST-DUP FILE3 is true for JOB iteration 12.

The FIRST-DUP and LAST-DUP conditions are also DUPLICATE conditions. A record that satisfies the IF LAST-DUP or IF FIRST-DUP condition also satisfies the IF DUPLICATE condition.

Refer to the *CA-Easytrieve Language Reference Guide* for more detailed examples of conditional expressions.

Updating a Master File

The next example illustrates updating a master file based upon matching transaction file records. The program assumes:

- A new master record is written when a match exists between the master file and the transaction file.
- There should be no duplicate transactions for a given master record. If this occurs, the first duplicate is processed but subsequent duplicates are bypassed.
- No transaction records should exist without a matching master record. If this occurs, the record is displayed on an error report and processing is bypassed.

```

FILE OLDMSTR SEQUENTIAL
O-KEY  1 2 N
O-AMT  3 3 N
FILE TRANS  SEQUENTIAL
T-KEY  1 2 N
T-AMT  3 3 N
FILE NEWMSTR SEQUENTIAL
N-KEY  1 2 N
N-AMT  3 3 N
JOB INPUT (OLDMSTR KEY(O-KEY) +
           TRANS KEY(T-KEY)) NAME MYPROG
* FOR MATCHED: UPDATE WITH TRAN AMT AND PUT NEWMSTR.
* IF TRAN IS A DUPLICATE BUT NOT THE FIRST, BYPASS THE RECORD.
IF MATCHED
  IF DUPLICATE TRANS AND NOT FIRST-DUP TRANS
    GOTO JOB
  END-IF
  MOVE O-KEY TO N-KEY
  N-AMT = O-AMT + T-AMT
  PUT NEWMSTR
  GOTO JOB
END-IF
* ON OLDMSTR ONLY: PUT THE NEWMSTR WITHOUT ANY UPDATE.
IF OLDMSTR
  PUT NEWMSTR FROM OLDMSTR
  GOTO JOB
END-IF
* ON TRANS ONLY: PRINT ERROR REPORT.
IF TRANS
  PRINT ERROR-RPT
  GOTO JOB
END-IF
*
REPORT ERROR-RPT
  TITLE 'REPORT OF TRANSACTION WITH INVALID KEYS'
  LINE T-KEY T-AMT

```

Single File Keyed Processing

Using Synchronized File Processing on a single file allows you to compare the contents of a key field or fields from one record to the next and use IF tests to group records according to the key fields. The file name is coded on the JOB INPUT statement as follows:

```
JOB INPUT (filename KEY (keyfield...))
```

Using single file input allows you to determine the start of a new key value and the end of the current key value by use of IF tests.

The following IF statement determines the start of a new key:

```
IF FIRST-DUP filename OR NOT DUPLICATE filename
```

The next IF statement determines the end of the current key:

```
IF LAST-DUP filename OR NOT DUPLICATE filename
```

The file must be in ascending order by its key value(s).

PRINTER Files

CA-Easytrieve enables you to create files with a device type of PRINTER to write printed output with the REPORT and/or DISPLAY statements. See Routing Printer Output in the “Report Processing” chapter for more information on output creation. A PRINTER file can be directed to one of the following destinations:

- The user’s terminal
- Any other terminal (only in CICS)
- Your operating system’s spooling subsystem
- An external data set.

The destination is determined by the FILE statement for the PRINTER file.

Defining a PRINTER File

A PRINTER file is specified by coding PRINTER as the device type on the FILE statement. You designate the destination of the file by specifying the data set type. The data set type can be one of the following:

- TERMINAL indicates that the output for the printer is routed to a terminal. The terminal can be a display terminal or an online CICS printer terminal. In CICS, you can specify the ID of a terminal other than the originating user’s terminal. When you do not specify a terminal ID (regardless of operating environment), the output is routed back to the originating terminal.
- SPOOL indicates that the output for this printer is routed to the operating system spooling subsystem. You can also specify the output class, destination node, and destination userid. In CMS, spooling requires that the user’s machine has its printer spooled to RSCS.

- If you are not executing in a CICS environment, SYSNAME associates the PRINTER file to an external data set. If you do not specify a valid data set type for the PRINTER file, CA-Easytrieve attempts to write the output to an external data set. When SYSNAME is not coded for an external data set, the file name is used as the external name of the file.
- On the workstation, SYSNAME (or the file name) can specify a device name (LPT1, CON).

SYSPRINT

SYSPRINT is a system-defined PRINTER file where printed output is sent unless otherwise specified. You can override this default by specifying the name of your PRINTER file on the REPORT and/or DISPLAY statement. SYSPRINT is routed to the destination specified in your Site Options. This may cause the output to be sent to one of the above destinations. See your system administrator for more information.

Workstation Files

CA-Easytrieve can access most workstation file structures in a way that allows the program to be portable to the mainframe environment.

Note: To support existing CA-Easytrieve FILE statements, file information is merged from FILE statements residing in the source program **and** the CA-Easytrieve/Workstation File Control Table (FCT). See the File Control Table (FCT) in Chapter 2 of the *CA-Easytrieve/Workstation User Guide* for more information.

Coding FILE Statements

The following guidelines apply when coding a workstation FILE statement.

SYSTEM Parameter

You must use the SYSTEM parameter on the FILE statement to describe operating environment-specific information about a file. This information may be necessary for CA-Easytrieve to process the file on the workstation. The SYSTEM parameter is ignored on the mainframe.

Note: CA-Easytrieve Plus versions 6.0 and below do not ignore the SYSTEM parameter and flag it as an error. To ensure portability to these versions, code the system information in an FCT entry.

Use the ACCESS subparameter to specify workstation access methods.

When you code a FILE statement for a file you want to access on both the workstation and mainframe, use the PATH subparameter instead of including the path on the SYSNAME parameter.

File Type

The FILE statement for each file specifies the type of CA-Easytrieve access. SEQUENTIAL is the default if not specified. Optionally, you can specify INDEXED or RELATIVE for keyed file access.

Note: CA-Easytrieve Plus batch versions do not support the SEQUENTIAL, INDEXED, and RELATIVE keywords. For compatibility, both CA-Easytrieve/Online and CA-Easytrieve/Workstation support older FILE statements (VS or unspecified file types). See Appendix C in the *CA-Easytrieve Language Reference Guide* for differences.

Record Format

Records can have either a fixed or variable format on the workstation.

ASCII variable-length records are always delimited by a carriage return (x'0D') and line feed character (x'0A') at the end of each record. EBCDIC variable-length records are delimited by the 2-byte user option VAREVAL. See the *User Guide* for option descriptions.

The record length of an EBCDIC variable-length file is the maximum data length plus four (4) bytes. Unlike the mainframe, the PC does not support a record-descriptor-word (RDW). For code portability, these four bytes are maintained internally, with the first two bytes being VAREVAL.

CA-Easytrieve/Workstation reads a record until either a delimiter is encountered or until the specified record length is read. If the delimiter characters are not located, an error message is issued and the program terminates.

When creating a variable-length file, assign the length of the record to the RECORD-LENGTH field before a PUT or WRITE statement is executed. If you do not do this, the record is padded with spaces up to the specified record length or default record length (if not specified).

Note: Use caution when specifying variable-length EBCDIC files. There can be cases in which the EBCDIC delimiter characters appear as part of the legitimate data. CA-Easytrieve/Workstation interprets these as the end of record delimiter, which does not produce the results expected.

Note: BTRIEVE variable-length records are not supported.

Fixed-length records are logical in nature. The record length of the file determines the end of one record and the beginning of the next.

Logical Record Length

If you do not specify the logical record length (LRECL) of the file, CA-Easytrieve/Workstation assumes the record length is 256 bytes. If the file is variable EBCDIC, this implies 252 bytes of data and four bytes for the RDW. For variable-length files (carriage-return/line-feed terminated ASCII), 256 bytes can be wasteful if the true record length is very small. If the true record length exceeds 256 bytes, CA-Easytrieve/Workstation issues an error message. For fixed format files, the length of fixed files is logical in nature. If set incorrectly, CA-Easytrieve/Workstation will not find fields where expected. Invalid data found in a numeric field could cause execution errors. CARD and PUNCH files are assumed to be 80 bytes.

File Code System

Use the CODE parameter of the file to specify the code system. CA-Easytrieve programs running on the workstation can access files using either ASCII or EBCDIC code systems. Code systems are converted as necessary. The code system defaults to the CODE PROCESS parameter of the PARM statement or system option. Some code systems are not supported in some access systems. CA-Easytrieve always assumes that PRINTER and variable-length files are ASCII.

Allowed Field Types

Refer to the following table for field types normally used for files:

Code System	Field Types
Variable-length ASCII	A, F
Fixed-length ASCII	A, F, I, S, D, B
EBCDIC	A, N, P, U, B

CA-Easytrieve always assumes that N, P, and U type numeric fields are EBCDIC numeric format.

Supported File Structures

The following list describes the supported file structures on the workstation and any special considerations for using them.

Sequential

File type: unspecified (Batch, Online, or UNIX portability) or SEQUENTIAL (Online or UNIX portability)

Record format: Fixed or variable

Code sets: ASCII or EBCDIC

SYSTEM ACCESS: not required

Indexed Sequential (FABS ISAM)

File type: IS (Batch or Online portability) or INDEXED (Online portability)

Record format: Fixed

Code sets: ASCII or EBCDIC

SYSTEM ACCESS: not required

Notes: The SYSTEM KEY parameter specifies the key of the record and is required. The SYSTEM INDEX parameter names the associated index component for the file. This allows a single data component to have alternate indexes or keys.

See the *CA-Easytrieve/Workstation User Guide* for information on a utility that can be used to build or rebuild the index component for ISAM files.

Relative (Random-access)

File type: VS (Batch or Online portability) or RELATIVE (Online or UNIX portability)

Record format: Fixed

Code sets: ASCII or EBCDIC

SYSTEM ACCESS: RRDS if VS

Notes: Random-access files are sequential files in which each record is located using a relative record number. When used in multi-user environments (LANs) for update the entire file remains locked while open.

BTRIEVE

File type: VS (Batch or Online portability) or INDEXED (Online portability)

Record format: Fixed

Code sets: ASCII or EBCDIC

SYSTEM ACCESS: BTRIEVE

Notes: The BTRIEVE record manager must be loaded when you execute programs accessing BTRIEVE files. See the *CA-Easytrieve/Workstation User Guide* for more information.

Use the SYSTEM ACCESS-PATH parameter to specify the access path used for record retrieval. Use the SYSTEM CREATE-PATH parameter to specify the access paths and keys to create. The keys need to be specified only during file creation. Use the SYSTEM OWNER parameter to specify the owner identification for the file. Use the SYSTEM CREATE-MODE parameter to set access rights and encryption mode for the file. Use the SYSTEM PAGESIZE parameter to set the data page size used by BTRIEVE. See your BTRIEVE documentation for complete information.

DBASE (xBASE)

File type: unspecified (Batch, Online, or UNIX portability) or SEQUENTIAL (Online or UNIX portability)

Record format: Fixed

Code sets: ASCII only

SYSTEM ACCESS: DBASE

Notes: Field definitions for the CA-Easytrieve file must be compatible with the dBASE structure. dBASE does not support N, P, B, U, I, S, or D field types. If used during output, CA-Easytrieve converts data in these field types to character fields. If used during input, CA-Easytrieve does not perform any conversion, enabling the character data to be defined by these field types.

Do not include the dBASE delete indicator byte present in every dBASE file in your record length.

When defining a file for output in dBASE format, the fields must be defined so that there are no undefined bytes between fields. For the dBASE software to correctly read the file, each field must be defined so that its first byte immediately follows the last byte of the preceding field.

Also, the record length must be equal to the sum of all field lengths defined in the dBASE output file. Note the following examples:

```
FILE DBASE-OUT F(100) +
  SYSTEM(ACCESS DBASE PATH 'DBFILE.DBF')
FLD-1  1 10 A
FLD-2 11 20 A
      * The record length is too long and
      * should be changed to F(30).
```

```
FILE DBASE-OUT F(35) +
  SYSTEM(ACCESS DBASE PATH 'DBFILE.DBF')
FLD-1  1 10 A
FLD-2 11 20 A
      * FLD-2 should be defined so that its
      * starting position is 11, not 16.
      * Leaving a blank space between fields
      * causes errors when the file is read
      * into dBASE. When you change the
      * definition for FLD-2 to FLD-2 11 20 A,
      * you should also change the record
      * length to F(30).
```

LOTUS

File type: unspecified (Batch, Online, or UNIX portability) or SEQUENTIAL (Online or UNIX portability)

Record format: Fixed

Code sets: ASCII only

SYSTEM ACCESS: LOTUS

Notes: CA-Easytrieve supports both WKS and WK1 file formats. The format is determined from the extension of the file. If not extension is specified, WKS is assumed.

When used as input, CA-Easytrieve ignores titles and formulas in the spreadsheet. When processing each row, CA-Easytrieve assigns each cell, left-to-right, into each base field defined in the CA-Easytrieve file. Data in the cell must conform to the field type and data is either truncated or padded to fit the field.

Numeric data in the spreadsheet is converted to the correct format if defined as F, N, P, B, U, I, S, or D.

When used as output, CA-Easytrieve creates cells, left-to-right, from each base field. Cell size is set to the length of the CA-Easytrieve field. F, N, P, B, U, I, S, or D type data is converted to a numeric cell when written to the spreadsheet.

Note the following example:

```
* This program is an example of how to read a
* Lotus 'WK1' format file as sequential input:
* as you can see, the only major difference
* between processing a Lotus file and a standard
* 'flat' sequential file is in the access method
* that is specified on the file statement.
```

* Note: The file 'PERSNL.WK1' is distributed
 * with CA-Easytrieve/Workstation and is normally
 * loaded into the 'SAMPLES' subdirectory of the
 * product whenever the 'INCLUDE SAMPLE FILES'
 * option of the installation is chosen.

```

PARM CODE PROCESS ASCII
FILE LOTUS-IN  F(75)      +
                SYSTEM(PC PATH('PERSNL.WK1') ACCESS LOTUS )
                NAME      1      16  A
                ADDR      *      20  A
                CITY      *      13  A
                ST        *       2  A
                ZIP       *       6  F
                PAY1      *       9  F  2
                PAY2      *       9  F  2

JOB INPUT LOTUS-IN

                PRINT REPT1

REPORT REPT1   SPACE 1
CONTROL
TITLE 1 'TEST REPORT USING A LOTUS FILE AS +
        INPUT'
LINE NAME ADDR CITY ST ZIP PAY1 PAY2

```

CA-SuperCalc

File type: unspecified (Batch, Online, or UNIX portability) or SEQUENTIAL (Online or UNIX portability)

Record format: Fixed

Code sets: ASCII only

SYSTEM ACCESS: SUPERCALC

Notes: CA-Easytrieve supports any CA-SuperCalc version 5 file.

When used as input, CA-Easytrieve ignores titles in the spreadsheet. When processing each row, CA-Easytrieve assigns each cell, left-to-right, into each base field defined in the CA-Easytrieve file. Data in the cell must conform to the field type and data is either truncated or padded to fit the field. Numeric data in the spreadsheet is converted to the correct format if defined as F, N, P, B, U, I, S, or D.

When used as output, CA-Easytrieve creates cells, left-to-right, from each base field. Cell size is set to the length of the CA-Easytrieve field. F, N, P, B, U, I, S, or D type data is converted to a numeric cell when written to the spreadsheet.

Comma-Delimited

File type: unspecified (Batch, Online, or UNIX portability) or SEQUENTIAL (Online or UNIX portability)

Record format: Variable

Code sets: ASCII only

SYSTEM ACCESS: DELIMITED

Notes: When processing each record for input, CA-Easytrieve assigns each field, left-to-right, into each base field defined in the CA-Easytrieve file. Data in the field must conform to the field type and data is either truncated or padded to fit the field. F type data in the file is converted to the correct format if defined as N, P, B, U, I, S, or D. Null alphanumeric values assigned into a varying alphanumeric field have a length of zero.

When used as output, CA-Easytrieve creates record fields, left-to-right, from each base field. Field size is set to the length of the CA-Easytrieve field. N, P, B, U, I, S, or D type data is converted to F type when written. Varying length alphanumeric fields with a length of zero create a null alphanumeric value.

Host Mainframe

File type: unspecified (Batch or Online portability) or SEQUENTIAL (Online portability)

Record format: Fixed

Code sets: EBCDIC

SYSTEM ACCESS: HOST

Notes: The host file is accessed using the file transfer facility of the High level Applications Level Interface (HLLAPI). Requirements for this feature are:

- An IBM PC 3270 connection to the mainframe.
- A PC 3270 emulator program that supports HLLAPI. See the *CA-Easytrieve/Workstation User Guide* for information on HLLAPI requirements and setup options.
- HLLAPI executing resident on the workstation.
- The IND\$FILE program executable in TSO or CMS.

When CA-Easytrieve opens a file for input that specifies HOST, a file transfer request is made to download the file to a system-defined virtual file on the workstation. When the transfer is complete, CA-Easytrieve processes the file. CA-Easytrieve assumes the file is in EBCDIC format on the mainframe. If you specify CODE ASCII on the FILE statement, CA-Easytrieve instructs the file transfer to convert the code system of the file from EBCDIC to ASCII and add carriage return and line feed characters (if variable) during the transfer.

When CA-Easytrieve opens a file for output that specifies `HOST`, output records are written to a virtual file on the workstation. When the file is closed, the virtual file is uploaded using a file transfer request to the mainframe. CA-Easytrieve assumes the file is in EBCDIC format on the mainframe. If you specify `CODE ASCII` on the `FILE` statement, CA-Easytrieve instructs the file transfer to convert the code system of the file from ASCII to EBCDIC and remove carriage return and line feed characters (if variable) during the transfer.

UNIX Files

The following guidelines apply when you code the `FILE` statement in UNIX.

File Type

The `FILE` statement for each file specifies the type of CA-Easytrieve access. CA-Easytrieve in the UNIX environment supports `SEQUENTIAL`, `INDEXED` and `RELATIVE` file types. `SEQUENTIAL` is the default if not specified. `INDEXED` files can be processed using various access methods. See *Executing Your Program in UNIX* in the *User Guide*.

Record Format

`RELATIVE` and `INDEXED` files permit only fixed format records. `SEQUENTIAL` files can have fixed or variable format records. Variable format records are line-feed (newline) delimited. Only text files can be variable. Binary data cannot be stored in a sequential variable file. You should set your record length to the length of the longest text. CA-Easytrieve reads the file for record-length-plus-one characters or up to a newline character, whichever comes first. If a newline character is not present, CA-Easytrieve issues an I/O error.

Fixed format records are fixed in length. The record length of the file determines where each record in the file begins. For example, if a file has a record length of 20, bytes 0 through 19 make up the first record, bytes 20 through 39 make up the second record, and so on. You can store binary data in fixed format files. Because fixed format records on UNIX and the Workstation are not line-feed delimited, you can get unpredictable results viewing or editing them.

Logical Record Length

You should specify the logical record length for each file. The defaults CA-Easytrieve may assume in the UNIX environment most likely are wrong.

- If you set the record length of a variable file too short, an error occurs.

- If you set the record length of a fixed file incorrectly, any record after the first appears to have fields shifted out of place. Often, this causes data errors.

Generally, CA-Easytrieve computes the default record length to be the greater of the following items:

- The highest location of a field defined in the file.
- The WORKAREA parameter of the FILE statement.
- The logical record length specified on the FILE statement.

C-ISAM

Creating C-ISAM files requires that CA-Easytrieve know what field represents the key of the record. See the FILE statement KEY parameter for details. CA-Easytrieve receives key information from C-ISAM after the files are created. Currently, CA-Easytrieve supports only fixed-length, unique single-keyed C-ISAM files and only as INDEXED files.

Not all C-ISAM data types are directly supported by CA-Easytrieve. The table below shows the C-ISAM data types and their CA-Easytrieve equivalents.

C-ISAM Data Type	CA-Easytrieve Field Definition
CHARTYPE	x A
INTTYPE	2 I Ø
LONGTYPE	4 I Ø
FLOATTYPE	4 A*
DOUBLETTYPE	8 A*
DECIMALTYPE	x A*

*These C-ISAM numeric data types are not directly usable within CA-Easytrieve. C-ISAM provides functions to convert between their machine-independent data representation and the internal representation required by CA-Easytrieve when it executes. You can call these routines from a FILE MODIFY exit to reformat data into any of the various CA-Easytrieve numeric data types. A sample exit is supplied with CA-Easytrieve as CISAMXIT.c.

C-ISAM execution requires special execution setup. See the *User Guide* for details on executing your C-ISAM program.

Overview

CA-Easytrieve provides optional processing facilities for SQL databases. The following SQL databases are supported:

- IBM's DB2, version 2.2 and greater
- IBM's DB2 for AIX, version 2.1 and greater
- IBM's SQL/DS, version 2.2 and greater
- CA-Datacom/PC, version 1.0 and greater
- CA-Datacom/DB with SQL, version 8.0 and greater
- CA-IDMS, version 12.0 and greater
- CA-Ingres, HP-UX and AIX, version 6.4 and greater
- CA-OpenIngres, HP-UX, and AIX, version 1.0 and greater.
- ORACLE, HP-UX and AIX, version 7.1 and greater.

Mainframe users:

- Before CA-Easytrieve can process these databases, the CA-Pan/SQL Interface product, version 2.4, must be correctly installed. See the *CA-Pan/SQL SQL Interface Installation Guide* for complete information.

Workstation users:

- CICSSERV.COM must be loaded into memory before CA-Easytrieve can process queries to CA-Datacom/PC. If CICSSERV.COM is not loaded into memory before compiling an SQL CA-Easytrieve program, the compile process suspends processing on the system. See the *CA-Datacom/PC MS-DOS Database and System Administration Guide* for more information about the CICSSERV facility.

UNIX users:

- All CA-Pan/SQL Interface functionality is installed with CA-Easytrieve. No additional installation or documentation is required.

To use these facilities effectively, you should have a basic knowledge of SQL and the given database management system to be processed.

Programming Methods

There are two programming methods supported for processing SQL databases:

- Using native SQL statements to manually manage the SQL cursor
- Allowing CA-Easytrieve to automatically manage the SQL cursor.

Native SQL Statements

CA-Easytrieve supports most of the SQL statements available for a given DBMS. The exceptions are those statements that are compiler directives and statements that cannot be dynamically “prepared.” Using these native SQL statements, you can code fully SQL-compliant programs in which you control the SQL cursor operation. All native SQL statements are prefixed with the SQL keyword. See the *CA-Easytrieve Language Reference Guide* for complete syntax. A list of all supported and unsupported SQL commands is provided later in this chapter.

Automatic Cursor Management

There are two ways that CA-Easytrieve can manage the SQL cursor for you:

- CA-Easytrieve files
- Automatic retrieval without a file.

CA-Easytrieve Files

CA-Easytrieve can automate SQL cursor management when you associate an SQL cursor with a CA-Easytrieve file. The SQL file can then be accessed in two ways:

- JOB INPUT statement - With each iteration of the JOB statement or activity, another row from the table is automatically retrieved into the file’s data area. Even if you have only a basic knowledge of SQL, you can report on data contained in an SQL database.
- CA-Easytrieve SQL-like statements - Use the following statements to read and write SQL data on a controlled basis:

CLOSE	DELETE
FETCH	INSERT
SELECT	UPDATE.

Automatic Retrieval without a File

Automatic retrieval does not require that you define a CA-Easytrieve file. In this read-only method, SQL must be coded on the JOB statement in place of a file name. A SELECT statement must be coded directly after the JOB statement to specify the columns to be retrieved and the host variables to receive the data. Each time the JOB activity is iterated, another row of SQL data is retrieved. This is a simple way to retrieve SQL data into working storage or into an extract file for subsequent output.

CA-Easytrieve SQL Statement Rules

There are several differences in the rules to follow when coding SQL control statements in CA-Easytrieve. The following syntax rules apply:

- Operators must be separated by blanks.
- Standard CA-Easytrieve continuation conventions are followed.
- Commas are not ignored.
- The period is used as a qualification separator, not to signify end-of-statement.
- The colon is used to identify host/indicator variables, not as a qualification separator.
- An SQL statement cannot be followed by another statement on the same source record.

Program Environment

CA-Easytrieve processes SQL statements using the CA-Pan/SQL Interface product on the mainframe and internal interfaces in UNIX and on the Workstation. A specific implementation exists for each supported database:

- For mainframe DB2, a dynamic and static interface are supported.
- For SQL/DS, “extended dynamic” SQL is supported. SQL statements are dynamically prepared during the compilation of your CA-Easytrieve program and an access module or package is created. At runtime, SQL statements are executed from the access module or package.
- The CA-Datacom/DB SQL interface is very similar to the SQL/DS interface. An access plan is created at compile time from which SQL statements are executed.
- The CA-Datacom/PC interface is similar to the CA-Datacom/DB SQL interface on the mainframe. An access plan, from which SQL statements are executed, is created at compile time .

- The CA-IDMS SQL interface is strictly a dynamic interface for both compilation and execution.
- The UNIX CA-Ingres, DB2, and ORACLE SQL interfaces are strictly dynamic interfaces for both compilation and execution.

See Unsupported SQL Commands later in this chapter, for a list of commands that cannot be coded in CA-Easytrieve programs.

Units of Work

Each CA-Easytrieve activity is considered a separate SQL unit of work or transaction. If COMMIT TERMINAL is specified on the activity statement, a commit is automatically executed during terminal I/O. If COMMIT ACTIVITY is specified in the activity, a commit is also executed following the termination of the activity. A ROLLBACK statement is automatically executed if CA-Easytrieve detects an error condition or if you code a STOP EXECUTE statement in your program.

You can issue your own COMMIT and ROLLBACK statements to signal the successful or unsuccessful end of the transaction. These COMMIT and ROLLBACK statements are performed in addition to the ones CA-Easytrieve does automatically.

Each time a COMMIT or ROLLBACK statement is executed, all open SQL cursors are closed. See the “Coding a CA-Easytrieve Program” chapter for more information on commit processing. Exceptions may exist for specific databases that maintain cursor positioning across commits or syncpoints.

PARM Statement Parameters

The following PARM statement parameters set the SQL environment for the program:

Database	Parameters that Set Environment
DB2	SQLID (mainframe only) SSID PLAN (mainframe only) BIND (mainframe only) SQLSYNTAX
SQL/DS	USERID PREPNAME SQLSYNTAX
CA-Datacom/PC	PREPNAME

CA-Datacom/DB	PLANOPTS PREPNAME SQLSYNTAX
CA-IDMS	USERID SQLSYNTAX
CA-Ingres	SSID USERID SQLSYNTAX
ORACLE	USERID SQLSYNTAX

In all environments, use the `SQLSYNTAX` parameter to specify the level of SQL syntax checking to be performed on the SQL statements in your program. `SQLSYNTAX FULL` specifies that SQL statements will undergo detail-level syntax checking. An SQL `PREPARE` statement is executed for those SQL statements that can be dynamically prepared. Your DBMS must be available to CA-Easytrieve. `SQLSYNTAX PARTIAL` indicates that your SQL statements are checked for valid commands and secondary keywords. No connection is made to your DBMS unless you have an SQL `INCLUDE` statement in your program. Your program will not execute until it has undergone `FULL` syntax checking.

You can use the `SQLSYNTAX NONE` parameter on the `PARM` statement with a static bind if you want syntax checking to be performed by the DB2 preprocessor in a batch environment. An option of `NONE` causes your program to undergo “partial” syntax checking. If no partial level compile errors are found and a `BIND` option of `STATIC-ONLY` is specified, and no other non-SQL syntax errors are found, your program continues to execution.

DB2

The `SQLID` parameter of the `PARM` statement results in the execution of the SQL `SET CURRENT SQLID` command by the SQL Interface at compile time. The SQL `SET CURRENT SQLID` command is executed at runtime for controlled or automatic processing. Execution of the SQL `SET CURRENT SQLID` command is valid for sites that have an external security package that supports group IDs.

The `SSID` parameter of the `PARM` statement can be used to specify the desired DB2 subsystem in non-CICS environments. If the `SSID` parameter is not coded, the SQL interface gets the DB2 subsystem ID from the DB2 system default module `DSNHDECP`. `DSNHDECP` is made available through the `JOBLIB` or `steplib` libraries. In CICS, the currently attached subsystem is used. In UNIX, the `SSID` identifies the database to be connected.

Static SQL is used to improve the performance of an SQL program. In a static SQL program, all SQL statements are known ahead of time and an optimized plan is created prior to execution time.

Static SQL is specified by two parameters on the PARM statement. PLAN specifies the name of the DB2 static-command-program and its planname. The command program can either be linked with the CA-Easytrieve program or linked as a separate load module. A BIND parameter of STATIC-ONLY or ANY causes the static-command-program to be generated.

To run your program statically, you must run special steps to create and link the static command program and plan. See the “Batch Compilation” and “Link-Editing” chapters in the *CA-Easytrieve/Online User Guide* for a complete discussion.

Dynamic SQL is used to process your program when running under the interpreter, regardless of the BIND parameter specified.

Static SQL and the LINK Statement

When you execute multiple static SQL programs in a given transaction or task, you must bind all of the involved DBRMs into a single plan. Therefore, specify the same planname for the PLAN parameter of each application program.

SQL/DS

Specify the SQL/DS userid to be used for compiling the program on the USERID parameter of the PARM statement. At execution time, a CONNECT is executed by CA-Easytrieve for automatic processing only. For controlled processing, you must code the SQL CONNECT command providing the values for an *sqlid* and password.

Specify the name of the SQL/DS access module for this program on the PREPNAME parameter of the PARM statement. When an SQL program is compiled, an access module is created or replaced. You should use unique access module names for each application program to avoid using the default name. In a high volume system, using the default PREPNAME can result in catalog contention and a -911 SQLCODE resulting from a rollback.

CA-Datacom/PC

Specify the name of the CA-Datacom/PC PLAN for this program on the PREPNAME parameter of the PARM statement. When an SQL program is compiled, a PLAN is created or replaced. You should use unique PLAN names for each application program to avoid using the default PLAN name from the Site Options Table.

CA-Datcom/DB

Use the PLANOPTS parameter on the PARM statement to specify the name of a CA-Pan/SQL PLAN options module to override the default module (DQSMPLN@). See the *CA-Pan/SQL SQL Interface Installation Guide* for more information about defining your own options module.

CA-IDMS

Use the USERID parameter to specify the name of the IDMS dictionary to be used for explicit connect (no password is needed).

If no dictionary name is provided, an implicit connection is attempted when the first SQL statement is processed. The dictionary name is then obtained from the SYSCTL DD card provided in your JCL.

CA-Ingres

Use the SSID parameter on the PARM statement to specify the name of the database to which this session will connect.

Use the USERID parameter on the PARM statement to specify the user identifier under which this session will run. The password subparameter is ignored.

ORACLE

Specify the ORACLE userid to be used for compiling the program on the USERID parameter of the PARM statement. At execution time, a CONNECT is executed by CA-Easytrieve for automatic processing only. For controlled processing, you must code the SQL CONNECT command providing the values for an *sqlid* and password.

Library Section Definition

Before SQL data can be accessed, you must define the fields to hold the columns to be retrieved. These fields are known as host variables.

If you are using native SQL commands or using automatic retrieval without a file, you usually define the fields as working storage fields. Alternatively, you can define the fields within an active output file. This is an effective method to select SQL data into a sequential file for extraction purposes. You must specify which columns are to be retrieved and which host variables are to receive the data.

When using a CA-Easytrieve file, however, fields defined within the file correspond to the selected columns of the SQL table. The table columns are retrieved into the file fields.

SQL Catalog INCLUDE Facility

The SQL catalog INCLUDE facility can be used to automatically generate CA-Easytrieve field definitions directly from the SQL catalog. This eliminates the need to code host variable definitions in the library section of your program.

The SQL INCLUDE statement names the SQL table or view from which column names and data types are obtained, and defines the location at which the field definitions are generated.

The SQL INCLUDE statement must precede any other SQL or SELECT statements and must be coded in the library section of your CA-Easytrieve program.

Note: To use the SQL catalog INCLUDE facility on the workstation, your database administrator must have installed the CA-Easytrieve workstation compiler's plan and granted you access to the system catalog tables.

Note: To use the SQL catalog INCLUDE facility for ORACLE, your database administrator must have installed the catalog views (*catalog.sql*).

Processing NULLable Fields

CA-Easytrieve supports the SQL concept of a null data value. Null is a value that denotes the absence of a known value for a field. Specify the keyword NULLABLE on the SQL INCLUDE statement to generate the null indicator variables. CA-Easytrieve does the rest of the processing for you when processing the SQL table as a file.

Note: CA-Easytrieve/Workstation defines binary fields as either a B or I data type, depending on the SQL setting in the Site Options Table.

When a field is defined as nullable, you can use special processing statements:

- IF NULL can be used to determine if the field contains a null value.
- MOVE NULL can be used to set a field's value to null.

Manual NULL Processing

When you use native SQL statements or automatic retrieval without a file, you define null values differently. You define an indicator variable as a two-byte quantitative binary field (2 B 0). This indicator variable is then used in the INTO clause of the native or automatic SELECT statement. SQL returns a negative value to the indicator variable when the field's value is null. See the native SQL examples for the use of manual indicator values.

Note: CA-Datcom/PC uses I type fields as indicator variables. You can code B or I data types as indicator variables, however, if you code B type data, conversion is performed whenever the database is accessed.

SQL Data Types

The following tables illustrate SQL data types and corresponding CA-Easytrieve field definitions. SQL provides some data conversion in SQL assignments and comparisons. Refer to your SQL manuals for more information on SQL data conversions. See the corresponding notes after the tables for asterisked items in the tables.

The first table includes DB2, SQL/DS, CA-Datcom/DB, and CA-Datcom/PC databases.

SQL	CA-Easytrieve	DB2, SQL/DS	CA-Datcom/ DB SQL	CA-Datcom/ PC SQL
INTEGER	4 B 0 4 I 0	Y	Y	Y *4
SMALL INTEGER	2 B 0 2 I 0	Y	Y	Y *4
DECIMAL (x,y)	x P y	Y	Y	Y *5
UNSIGNED DECIMAL (x,y)	x P y	N	N	Y *5
CHARACTER (x)	x A	Y	Y	Y *6
VARCHAR (x)	x A VARYING (x <= 254)	Y	Y (8.1)	N
TEXT (x)	x A VARYING	N	N	N
LONG VARCHAR (x)	x A VARYING (x > 254)	Y	Y (8.1)	N
VARCHAR2 (x)	x A VARYING	N	N	N

SQL	CA-Easytrieve	DB2, SQL/DS	CA-Datacom/ DB SQL	CA-Datacom/ PC SQL
RAW	x A VARYING	N	N	N
LONG RAW (x)	x A VARYING	N	N	N
NUMERIC (x,y)	x N y	N	Y	Y *7
UNSIGNED NUMERIC (x,y)	x N y	N	N	Y *7
FLOAT	10 P 3 *1	Y	Y	Y
REAL	10 P 3 *1	Y	Y	Y
DOUBLE PRECISION	10 P 3 *1	Y	Y	Y
NUMBER	x P y	N	N	N
MONEY	10 P 2	N	N	N
GRAPHIC (x)	x M x K	Y	N	N
VARGRAPHIC (x)	x M VARYING x K VARYING (x <= 254)	Y	N	N
LONG VARGRAPHIC (x)	x M VARYING x K VARYING (x > 254)	Y	N	N
DATE	10 A *8	Y	Y	Y
TIME	8 A	Y	Y	Y
TIMESTAMP	26 A	Y	Y	Y
LONG INTEGER	8 B O *2	N	N	N
BINARY	x B y *3	N	N	N
none	x U y	-	-	-

The next table includes CA-IDMS, CA-Ingres, and ORACLE databases.

SQL	CA-Easytrieve	CA-IDMS SQL	CA-Ingres	ORACLE
INTEGER	4 B 0 4 I 0	Y	Y	N
SMALL INTEGER	2 B 0 2 I 0	Y	Y	N

SQL	CA-Easytrieve	CA-IDMS SQL	CA-Ingres	ORACLE
DECIMAL (x,y)	x P y	Y	N	N
UNSIGNED DECIMAL (x,y)	x P y	Y	N	N
CHARACTER (x)	x A	Y	Y	Y
VARCHAR (x)	x A VARYING (x <= 254)	Y	Y	Y
TEXT (x)	x A VARYING	N	Y	N
LONG VARCHAR (x)	x A VARYING (x > 254)	Y	N	N
VARCHAR2 (x)	x A VARYING	N	N	Y
RAW (x)	x A VARYING	N	N	Y
LONG RAW (x)	x A VARYING	N	N	Y
NUMERIC (x,y)	x N y	Y	N	N
UNSIGNED NUMERIC (x,y)	x N y	Y	N	N
FLOAT	10 P 3 *1	Y	Y	Y
REAL	10 P 3 *1	Y	N	N
DOUBLE PRECISION	10 P 3 *1	Y	N	N
NUMBER (x,y)	x P y *9	N	N	Y
MONEY	10 P 2	N	Y	N
GRAPHIC (x)	x M x K	Y	N	N
VARGRAPHIC (x)	x M VARYING x K VARYING (x <= 254)	Y	N	N
LONG VARGRAPHIC (x)	x M VARYING x K VARYING (x > 254)	Y	N	N
DATE	10 A *8	Y	Y	Y
TIME	8 A	Y	N	N
TIMESTAMP	26 A	Y	N	N
LONG INTEGER	8 B 0 *2	Y	N	N
BINARY	x B y *3	Y	N	N

SQL	CA-Easytrieve	CA-IDMS SQL	CA-Ingres	ORACLE
none	x U y	-	-	-

Note 1: On the Workstation, these types are converted to the CA-Easytrieve data definition of 8 D 0.

Note 2: LONG INTEGER is valid only for CA-Easytrieve/Workstation. When processing an SQL INCLUDE statement on the mainframe, this data type is converted to the CA-Easytrieve data definition of 10 P 0.

Note 3: The CA-Easytrieve data type of BINARY for a length other than 2 or 4 is valid only for CA-Easytrieve/Workstation. When processing an SQL INCLUDE statement on the mainframe, this data type is converted to x A.

Note 4: Data conversion required.

Note 5: Data conversion required for EBCDIC.

Note 6: Maximum length = 8 bytes.

Note 7: Maximum length = 15 bytes.

Note 8: For CA-Ingres, dates are 11 bytes in length.

Note 9: Maximum length = 10 bytes.

Decimal Data Types

For SQL DECIMAL data types, the scale is the same as the decimal places of a CA-Easytrieve field. SQL precision refers to the total number of digits that can occur in the packed field. A CA-Easytrieve length refers to the number of bytes occupied by the packed field. A CA-Easytrieve field that is 5 P 2 is the equivalent of an SQL DECIMAL data type of precision = 9 and scale = 2. Depending on your SQL release, SQL may not support CA-Easytrieve packed fields with lengths > 8.

SQL Syntax Checking

When an SQL statement is passed to SQL for syntax checking, host variables are converted to question marks (?) by CA-Easytrieve. It is possible that when an SQL error is detected, the question mark is identified as the field in error. In this case, you are responsible for looking up the error message and identifying which host variable is in error. Because host variables are replaced with question marks, their use in arithmetic expressions may result in compile errors. For DB2 and SQL/DS, an SQLCODE of -418 can occur.

System-Defined File Fields

When using a CA-Easytrieve file to process an SQL database, the following system-defined fields are used:

RECORD-COUNT

RECORD-COUNT contains the number of rows returned to the CA-Easytrieve program. This is the number of rows fetched either by automatic or controlled processing.

RECORD-LENGTH

RECORD-LENGTH is the length of the SQL file. The length is the sum of the maximum lengths of all fields in the file.

EOF Processing

When the end of the table(s) has been reached, either with automatic (JOB) or controlled (FETCH) processing, the file is marked EOF (end of file). In automatic processing, execution stops, and the FINISH procedure (if present) executes. In controlled processing, you can code file presence tests (IF EOF *file-name*) to determine whether an end of file condition exists.

SQL Communications Area Fields

All of the SQL Communication Area fields (SQLCA) are automatically created in S (static) working storage when any of the following occurs:

- The first SQL-managed file is encountered
- The first SQL INCLUDE statement is encountered
- The first native or controlled SQL statement is found
- The first JOB INPUT SQL statement is found.

The fields generated for SQLCA for the supported SQL database management systems are shown in the following exhibits.

SQL Communication Area Fields for DB2, SQL/DS, CA-Ingres, and ORACLE

DEFINE SQLCA	S	136	A
DEFINE SQLCAID	SQLCA	8	A
DEFINE SQLCABC	SQLCA +8	4	B 0
DEFINE SQLCODE	SQLCA +12	4	B 0
DEFINE SQLERRM	SQLCA +16	72	A
DEFINE SQLERRML	SQLCA +16	2	B 0
DEFINE SQLERRMC	SQLCA +18	70	A
DEFINE SQLERRP	SQLCA +88	8	A

```
DEFINE SQLERRD      SQLCA +96      4 B 0 OCCURS 6
DEFINE SQLWARN      SQLCA +120     8 A
DEFINE SQLWARN0     SQLCA +120     1 A
DEFINE SQLWARN1     SQLCA +121     1 A
DEFINE SQLWARN2     SQLCA +122     1 A
DEFINE SQLWARN3     SQLCA +123     1 A
DEFINE SQLWARN4     SQLCA +124     1 A
DEFINE SQLWARN5     SQLCA +125     1 A
DEFINE SQLWARN6     SQLCA +126     1 A
DEFINE SQLWARN7     SQLCA +127     1 A
DEFINE SQLWARN8     SQLCA +128     1 A
DEFINE SQLWARN9     SQLCA +129     1 A
DEFINE SQLWARNA     SQLCA +130     1 A
DEFINE SQLEXT       SQLCA +131     5 A
```

SQL Communication Area Fields for CA-Datcom/DB and CA-Datcom/PC

SQLCA	S	196	A	
SQLCA-EYE-CATCH	SQLCA	8	A	
SQLCAID	SQLCA	8	A	
SQLCA-LEN	SQLCA +8	4	B 0	
SQLCABC	SQLCA +8	4	B 0	
SQLCA-DB-VRS	SQLCA +12	2	A	
SQLCA-DB-RLS	SQLCA +14	2	A	
SQLCA-LUWID	SQLCA +16	8	A	
SQLCODE	SQLCA +24	4	B 0	
SQLCA-ERROR-INFO	SQLCA +28	82	A	
SQLCA-ERR-LEN	SQLCA +28	2	B 0	
SQLCA-ERR-MSG	SQLCA +30	80	A	
SQLERRM	SQLCA +28	72	A	
SQLERRML	SQLCA +28	2	B 0	
SQLERRMC	SQLCA +30	70	A	
SQLCA-ERROR-PGM	SQLCA +110	8	A	
SQLERRP	SQLCA +110	8	A	
SQLCA-FILLER-1	SQLCA +118	2	A	
SQLCA-ERROR-DATA	SQLCA +120	24	A	
SQLCA-DSFCODE	SQLCA +120	4	A	
SQLCA-INFCODE	SQLCA +124	4	B 0	
SQLCA-DBC CODE	SQLCA +128	4	A	
SQLCA-DBC CODE-EXT	SQLCA +128	2	A	
SQLCA-DBC CODE-INT	SQLCA +130	2	B 0	
SQLCA-MISC-CODE1	SQLCA +132	4	A	
SQLCA-MISC-CODE2	SQLCA +136	4	B 0	
SQLCA-MISC-CODE3	SQLCA +140	4	A	
SQLCA-WRN-AREA	SQLCA +144	8	A	
SQLCA-WARNING	SQLCA +144	1	A	OCCURS 8
SQLWARN	SQLCA +144	8	A	
SQLWARN0	SQLCA +144	1	A	
SQLWARN1	SQLCA +145	1	A	
SQLWARN2	SQLCA +146	1	A	
SQLWARN3	SQLCA +147	1	A	
SQLWARN4	SQLCA +148	1	A	
SQLWARN5	SQLCA +149	1	A	
SQLWARN6	SQLCA +150	1	A	
SQLWARN7	SQLCA +151	1	A	
SQLCA-PGM-NAME	SQLCA +152	8	A	
SQLCA-AUTHID	SQLCA +160	18	A	
SQLCA-PLAN-NAME	SQLCA +178	18	A	

SQL Communication Area Fields for CA-IDMS

SQLCA	S	344	A	
SQLCAID	SQLCA	8	A	
SQLCODE	SQLCA +8	4	B 0	
SQLCSID	SQLCA +12	4	B 0	OCCURS 2
SQLCERC	SQLCA +20	4	B 0	
SQLCNRP	SQLCA +28	4	B 0	
SQLCSER	SQLCA +36	4	B 0	
SQLCLNO	SQLCA +44	4	B 0	
SQLCMCT	SQLCA +48	4	B 0	
SQLCOPTS	SQLCA +52	4	B 0	
SQLCFJB	SQLCA +56	4	B 0	
SQLCPCID	SQLCA +60	4	B 0	
SQLCLCID	SQLCA +64	4	B 0	
SQLCERL	SQLCA +68	2	B 0	
SQLCERM	SQLCA +72	256	A	
SQLSTATE	SQLCA +328	5	A	

Sample Database

The following two tables are used for all the examples in this chapter.

```
TABLE: PERSONNEL
-----
COLUMNS: EMPNAME          WORKDEPT  EMPPHONE
-----
DATA:     NORIDGE DEBBIE      901      5001
          OSMON  SAMUEL      901      5004
          MILLER JOAN        950      6034
          EPERT  LINDA       950      null
          STRIDE ANN         901      null
          ROGERS PAT         921      2231

EMPNAME   - CHAR(20) (NOT NULL)
WORKDEPT  - DECIMAL(3,0) (NOT NULL)
EMPPHONE  - DECIMAL(5,0) (NULL)
```

```
.....
TABLE: DEPARTMENTS
-----
COLUMNS: DEPTNAME          DEPTNUMBER
-----
DATA:     SHIPPING          901
          HUMAN RESOURCES    921
          ACCOUNTING         950
          DATA PROCESSING    951

DEPTNAME   - VARCHAR(20) (NOT NULL)
DEPTNUMBER - DECIMAL(3,0) (NOT NULL)
```

WORKDEPT in the PERSONNEL table corresponds with the DEPTNUMBER in the DEPARTMENTS table.

Working Storage Definitions

The next exhibit shows working storage field definitions for the sample tables in the previous exhibit.

```
DEFINE EMPNAME   W 20 A
DEFINE WORKDEPT  W 2 P 0
DEFINE EMPPHONE  W 3 P 0
DEFINE DEPTNAME  W 22 A VARYING
DEFINE DEPTNUMBER W 2 P 0
DEFINE NULLPHONE W 2 B 0 .* NULL INDICATOR
```

```
DEFINE USERID      W  8 A  VALUE('SQLDBA')    .* SQL/DS USERID
DEFINE PASSWORD    W  8 A  VALUE('SQLDBAPW')  .* SQL/DS PASSWORD
```

CA-Easytrieve SQL Files

In this method of processing SQL, CA-Easytrieve automates cursor management by associating an SQL cursor with a CA-Easytrieve file.

Processing Requirements

To process data from an SQL table using this method, you must code the following:

- A FILE statement specifying one or more table names. If all columns defined in the file are subject to update, specify the UPDATE keyword on the FILE statement.
- One or more field definitions for the columns within the table(s) that you want to retrieve. These definitions can be coded using the DEFINE statement or by using the SQL INCLUDE statement to automatically generate the definitions from the SQL catalog. If you want to selectively update only a few columns of those defined within the file, omit the UPDATE keyword from the FILE statement and specify UPDATE only on the field definitions (columns) you want to update. Coding UPDATE on the SQL INCLUDE statement causes generated definitions to be subject to update.
- A SELECT statement that defines the result set for the cursor. If you do not use a SELECT statement, CA-Easytrieve generates a default SELECT to retrieve all rows for the table(s). You can override this default by specifying your own file-based SELECT statement as the first statement following the JOB statement.

Overriding the default SELECT allows you to use SQL techniques to customize the result set for the cursor. For example, you can:

- Limit the result set to a subset of records (WHERE)
- Organize the data in the table by groups (GROUP BY)
- Limit the groups returned (HAVING)
- Sequence the returning records (ORDER BY).

In addition to overriding the default SELECT, you can code one or more SELECTS anywhere in your JOB activity. This enables you to use conditional logic to dynamically determine which SELECT is executed.

A SELECT statement for an SQL file is similar to opening the file. SELECTing a file that is already open first closes the file and then reopens the file based on the new SELECT statement.

See the *CA-Easytrieve Language Reference Guide* for more information on SELECT usage.

Operation

If you are executing in an SQL/DS, CA-Ingres, UNIX DB2, or ORACLE system, CA-Easytrieve generates and executes a CONNECT statement. You need not code an SQL CONNECT statement when using CA-Easytrieve automatic processing. The user ID and password parameters are taken from those specified in the USERID parameter of the PARM statement.

CA-Easytrieve checks the SQLCODE field following each execution of a file-based SQL statement. If the SQLCODE indicates an error, CA-Easytrieve issues an error message based on the SQL error and terminates execution. During automatic processing, an SQLCODE 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. During controlled processing, an SQLCODE indicating end of data causes CA-Easytrieve to set the value of EOF to true.

The SQL cursor that is automatically defined by a SELECT statement is closed following the termination of the activity that opened it.

Note: UNIX DB2 and ORACLE systems are limited to a maximum of 10 cursors.

Input Processing

You can access SQL data using a CA-Easytrieve file either automatically or using controlled processing.

Automatic Processing

CA-Easytrieve automatically accesses your SQL database if you name the SQL file as the input file on a JOB statement. When you specify an SQL file as automatic input, CA-Easytrieve prepares a default SELECT statement for the cursor:

The following program displays all records in the table, PERSONNEL:

```
FILE PERSNL SQL (PERSONNEL)
EMPNAME      * 20 A
WORKDEPT     *  2 P 0
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL
```

```
DISPLAY EMPNAME +2 WORKDEPT
```

You can override the default SELECT statement by coding a SELECT statement for the automatic input file as the first statement in the JOB, as shown in the following exhibit:

```
FILE PERSNL SQL (PERSONNEL)
EMPNAME * 20 A
WORKDEPT * 2 P 0
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL
  SELECT FROM PERSNL WHERE WORKDEPT = 901
  DISPLAY EMPNAME +2 WORKDEPT
```

The above program displays only the records for employees assigned to department 901. The SELECT statement provides the new default selection criteria.

Using DEFER with SELECT

You can use the DEFER parameter on the SQL FILE statement to delay SELECT processing. For example, assume you want to select only employee numbers in a particular department and the department number is kept in a card file. The SELECT statement contains a WHERE clause specifying the host variable in the card file. CA-Easytrieve opens the CARD file at the initiation of the JOB activity but the GET statement is coded in a START procedure. If the file SELECT is not DEFERred, the SELECT is performed using an uninitialized host variable. Coding DEFER enables the START procedure to get the card before the SELECT is performed.

In addition, if DEFER is not specified, the default SELECT is executed before the START procedure. Then the SELECT in the START procedure is executed. This causes extra processing that is not needed. An example follows:

```
FILE PERSNL SQL (PERSONNEL) DEFER
EMPNAME * 20 A
WORKDEPT * 2 P
FILE CARDFIL CARD
CARDDEPT 1 3 N
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL START GETCARD
  DISPLAY EMPNAME +2 WORKDEPT
  GETCARD. PROC
  GET CARDFIL
  SELECT FROM PERSNL WHERE WORKDEPT = :CARDDEPT
END-PROC
```

NULLable Field Processing

The following example illustrates how to process the nullable field, EMPPHONE. You must test for a nullable field before processing it. If EMPPHONE contains a null value, it is set to zero before displaying it:

```
FILE PERSNL SQL (PERSONNEL)
  SQL INCLUDE (EMPNAME, WORKDEPT, EMPPHONE) +
  FROM PERSONNEL LOCATION * NULLABLE
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL
  IF EMPPHONE NULL
    EMPPHONE = 0
  END-IF
  DISPLAY EMPNAME +2 WORKDEPT +2 EMPPHONE
```

Multiple Tables

The next example illustrates joining two tables, PERSONNEL and DEPARTMENTS, to report on employees and the departments in which they work:

```
FILE PERSNL SQL (PERSONNEL, DEPARTMENTS)
EMPNAME * 20 A HEADING 'EMPLOYEE NAME'
WORKDEPT * 2 P 0 HEADING ('DEPT', 'NO')
DEPTNAME * 22 A HEADING 'DEPARTMENT'
DEPTNUMBER * 2 P 0
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL
SELECT FROM PERSNL WHERE WORKDEPT = DEPTNUMBER
PRINT PERSNL-REPORT
REPORT PERSNL-REPORT
LINE EMPNAME WORKDEPT DEPTNAME
```

Both table names are specified on the FILE statement. The default SELECT is overridden because the result set should include only those DEPARTMENT records that match the department number of the PERSONNEL record.

Controlled Processing

You use the FETCH statement (in conjunction with SELECT and CLOSE statements) to retrieve the records from the file with controlled processing. You can code these statements within a PROGRAM, SCREEN, or JOB activity, with or without automatic input. The following rules apply:

- Controlled statements are not permitted in SORT or REPORT procedures.
- The FETCH statement cannot reference an automatic input file within the same JOB activity. You can FETCH from a file other than the file subject to automatic input.

In a PROGRAM Activity

Here is an example of controlled input from a default cursor:

```
FILE PERSNL SQL (PERSONNEL)
EMPNAME * 20 A
WORKDEPT * 2 P 0
PROGRAM NAME RETRIEVE-PERSONNEL
FETCH FROM PERSNL
DO WHILE NOT EOF PERSNL
  DISPLAY EMPNAME +2 WORKDEPT
  FETCH FROM PERSNL
END-DO
```

The PROGRAM activity fetches each row of the table and displays the fields. The DO statement executes until end-of-file.

In a JOB Activity

The same process used in a JOB activity is coded as follows:

```
FILE PERSNL SQL (PERSONNEL)
EMPNAME * 20 A
WORKDEPT * 2 P 0
```

```

JOB NAME RETRIEVE-PERSONNEL INPUT NULL
  FETCH FROM PERSNL
  IF NOT EOF PERSNL
    DISPLAY EMPNAME +2 WORKDEPT
  ELSE
    STOP
  END-IF

```

You must execute a STOP statement when end-of-file is reached because a NULL JOB activity automatically loops.

Random Processing

The following example illustrates a type of random processing in which the SQL file's cursor is built using a key contained in a sequential file.

```

FILE ESDS SEQUENTIAL
FILENAME 17 20 A
FILE PERSNL SQL (PERSONNEL) DEFER
EMPNAME * 20 A
WORKDEPT * 2 P 0
JOB NAME RETRIEVE-PERSONNEL INPUT ESDS
  SELECT FROM PERSNL WHERE EMPNAME = :FILENAME
  FETCH FROM PERSNL
  IF EOF PERSNL
    DISPLAY 'EMPLOYEE NOT ON FILE ' FILENAME
  ELSE
    DISPLAY EMPNAME +2 WORKDEPT
  END-IF
CLOSE PERSNL

```

The sequential file is read automatically by the JOB activity. For each record, a SELECT statement is executed to locate the employee in the PERSONNEL table. If the FETCH statement results in end-of-file, the employee is not found. Otherwise, the employee's name and department are displayed.

Synchronized File Processing

The following example illustrates a way to match a sequential file with an SQL file. This example uses the default SELECT and then matches the two files based on the employee name in both files:

```

FILE ESDS SEQUENTIAL
FILENAME 17 20 A
FILE PERSNL SQL (PERSONNEL)
EMPNAME * 20 A
WORKDEPT * 2 P 0
JOB NAME MATCH-PERSNL INPUT (ESDS KEY (FILENAME) PERSNL KEY (EMPNAME))
  IF NOT MATCHED AND ESDS
    DISPLAY 'EMPLOYEE NOT IN SQL FILE' FILENAME
  ELSE
    IF NOT MATCHED AND PERSNL
      DISPLAY 'EMPLOYEE NOT ON ESDS FILE' EMPNAME
    ELSE
      DISPLAY 'EMPLOYEE ON BOTH FILES' EMPNAME
    END-IF
  END-IF

```

The default SELECT could have been overridden by coding a SELECT as the first statement after the JOB statement.

Update Processing

Following are additional requirements for updating a CA-Easytrieve SQL file:

- You can name only one table on the FILE statement for updates, inserts, or deletions.
- You must specify the UPDATE parameter on the FILE statement if all fields defined are subject to update. If you want to update only certain fields, do not specify UPDATE on the FILE statement, but specify UPDATE on the DEFINE statement for each field to be updated. You can also use the SQL INCLUDE statement to automatically generate definitions with UPDATE.
- You must specify UPDATE on the FILE statement to insert or delete rows.
- For updating, you must code a SELECT statement for the file that contains the FOR UPDATE clause. If the FOR UPDATE clause is omitted, the first UPDATE statement is flagged in error. Default SELECT statements created by CA-Easytrieve do not contain the FOR UPDATE clause.
- For ORACLE, dynamic UPDATES and DELETES must be mimicked by CA-Easytrieve by selecting the ROWID of each row, then using that value to identify the current row during the UPDATE or DELETE. For this reason, you must use a SELECT statement with the FOR UPDATE clause.

Controlled Processing

The following example selects a specific row from the table, updates the department to 930, and moves a null data value to the phone number.

```
FILE PERSNL SQL (PERSONNEL) UPDATE
SQL INCLUDE (EMPNAME, WORKDEPT, EMPPHONE) +
FROM PERSONNEL LOCATION * NULLABLE
PROGRAM NAME RETRIEVE-PERSONNEL
SELECT FROM PERSNL WHERE EMPNAME = 'ROGERS PAT' FOR UPDATE
FETCH FROM PERSNL
IF EOF PERSNL
  DISPLAY 'EMPLOYEE NOT FOUND'
ELSE
  WORKDEPT = 930
  MOVE NULL TO EMPPHONE
  UPDATE PERSNL
END-IF
```

Automatic Processing

The following example changes the department number for all employees in department 901 to 921.

```
FILE PERSNL SQL (PERSONNEL) UPDATE
SQL INCLUDE (WORKDEPT) +
FROM PERSONNEL LOCATION *
JOB NAME RETRIEVE-PERSONNEL INPUT PERSNL
SELECT FROM PERSNL WHERE WORKDEPT = 901 FOR UPDATE
WORKDEPT = 921
UPDATE PERSNL
```

Deleting from an SQL File

The following example illustrates how to select a specific row from the table, and then delete it.

```
FILE PERSNL SQL (PERSONNEL) UPDATE
SQL INCLUDE (EMPNAME, WORKDEPT, EMPPHONE) +
  FROM PERSONNEL LOCATION * NULLABLE
PROGRAM NAME RETRIEVE-PERSONNEL
  SELECT FROM PERSNL WHERE EMPNAME = 'ROGERS PAT' FOR UPDATE
  FETCH FROM PERSNL
  IF EOF PERSNL
    DISPLAY 'EMPLOYEE NOT FOUND'
  ELSE
    DELETE FROM PERSNL
  END-IF
```

Inserting an SQL Row

The following example illustrates how to insert a new row into a table.

```
FILE PERSNL SQL (PERSONNEL) UPDATE
SQL INCLUDE (EMPNAME, WORKDEPT, EMPPHONE) +
  FROM PERSONNEL LOCATION * NULLABLE
PROGRAM NAME RETRIEVE-PERSONNEL
  EMPNAME = 'WIMN GLORIA'
  WORKDEPT = 921
  EMPPHONE = 3478
  INSERT INTO PERSNL
```

Automatic Retrieval without a File

In this method of processing, SQL data is retrieved as a result of a specially-coded JOB and SELECT statement combination. Automatic retrieval without a file is a read-only method that typically uses working storage fields as the receiving area for the data. This method allows some advanced selection techniques not available for cursors associated with CA-Easytrieve files.

Processing Requirements

To process data from an SQL table using this method, you must provide the following:

- One or more field definitions for the columns within the table(s) that you want to retrieve. These definitions can be coded using the DEFINE statement or by using the SQL INCLUDE statement to automatically generate the definitions from the SQL catalog. The definitions are usually coded in working storage, but you can define the fields in an output file for extraction purposes.
- A JOB statement with the INPUT SQL parameter. SQL signifies that the automatic processing does not involve a CA-Easytrieve file.

- A non-file-based SELECT statement that defines the result set for the cursor. Only one non-file based SELECT statement can be coded in each JOB activity.

This SELECT statement is very different from the file-based SELECT statement used with a CA-Easytrieve SQL file because it more closely approximates a true SQL SELECT clause. For example, you name the tables which participate in the result. Also, the SELECT can perform more advanced selections such as UNIONS.

Operation

If you are executing in an SQL/DS system, CA-Easytrieve generates and executes a CONNECT statement. You need not code an SQL CONNECT statement when using CA-Easytrieve automatic processing. The userid and password parameters are taken from those specified in the USERID parameter of the PARM statement.

CA-Easytrieve checks the SQLCODE field following each execution of the SELECT statement. If the SQLCODE indicates an error, CA-Easytrieve issues an error message based on the SQL error and terminates execution. An SQLCODE 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.

The SQL cursor that is automatically defined by a non-file based SELECT statement is closed following the termination of the JOB activity that opened it.

Retrieving All Columns

The code in the following exhibit retrieves all columns and all rows from the PERSONNEL table. A report is generated showing WORKDEPT, EMPNAME, and EMPPHONE. CA-Easytrieve sorts the report by WORKDEPT. Note the use of manual null variable indicators.

```

DEFINE EMPNAME      W 20 A
DEFINE WORKDEPT     W  2 P 0
DEFINE EMPPHONE     W  3 P 0
DEFINE NULLPHONE    W  2 B 0 .* NULL INDICATOR
JOB INPUT SQL NAME(SQLEX1)
  SELECT * FROM PERSONNEL
    INTO :EMPNAME, :WORKDEPT, :EMPPHONE :NULLPHONE
  IF NULLPHONE < 0 .* PHONE PRESENT?
    EMPPHONE = 0 .* NO, SET TO 0
  END-IF
  PRINT PERSNL
REPORT PERSNL LINESIZE 65
SEQUENCE WORKDEPT
LINE WORKDEPT EMPNAME EMPPHONE

```

Selected Columns

The code in the following exhibit retrieves all rows of selected columns from the PERSONNEL table and generates a report showing WORKDEPT and EMPNAME. SQL orders the rows by WORKDEPT.

```
DEFINE EMPNAME    W 20 A
DEFINE WORKDEPT   W  2 P 0
JOB INPUT SQL NAME(SQLEX2)
  SELECT EMPNAME, WORKDEPT      +
     FROM PERSONNEL             +
     ORDER BY WORKDEPT         +
     INTO :EMPNAME, :WORKDEPT
  PRINT PERSNL
REPORT PERSNL LINESIZE 65
  LINE WORKDEPT EMPNAME
```

Multiple Tables

The code in the following example retrieves an employee name and the corresponding department name from the PERSONNEL and DEPARTMENTS tables. This example shows parameters required for SQL/DS.

```
PARM USERID('SQLDBA' 'SQLDBAPW') +
  PREPNAME(EASYIOL 'SQLDBA')
DEFINE EMPNAME    W 20 A
DEFINE WORKDEPT   W  2 P 0
DEFINE EMPPHONE   W  3 P 0
DEFINE DEPTNAME   W 22 A    VARYING
DEFINE DEPTNUMBER W  2 P 0
DEFINE NULLPHONE  W  2 B 0  .* NULL INDICATOR
JOB INPUT SQL NAME(SQLEX3)
  SELECT EMPNAME, DEPTNAME      +
     FROM PERSONNEL, DEPARTMENTS +
     WHERE WORKDEPT = DEPTNUMBER +
     INTO :EMPNAME, :DEPTNAME
  PRINT PERSNL
REPORT PERSNL LINESIZE 65
  LINE EMPNAME DEPTNAME
```

Native SQL Processing

This method of processing uses native SQL statements that are equivalent to many of those used in COBOL. Using these native SQL statements, you can code fully SQL-compliant programs in which you control the SQL cursor operation. All native SQL statements are prefixed with the SQL keyword. See the *CA-Easytrieve Language Reference Guide* for complete syntax.

Processing Requirements

- The SQL DECLARE statement must be coded in the Library Definition section of a CA-Easytrieve program. All other SQL statements, except SQL INCLUDE, must be coded in the Activity Definition section.

- You should test the SQLCODE field in the SQLCA to determine whether or not the execution of each controlled processing statement is successful.

If the SQLCODE field contains a zero (0), you should test the SQLWARN0 field to ensure that no warning conditions were issued during processing of the SQL statement. Refer to the appropriate SQL reference manual to determine acceptable values for SQLWARN0.

- All SQL INCLUDE statements and SQL-managed file definitions must be coded prior to any controlled SQL statements.

Operation

Coding native SQL statements requires an advanced knowledge of SQL statements and of the database to be processed. Native SQL statements can be coded in any PROGRAM, SCREEN, or JOB activity. You cannot code them in SORT or REPORT procedures.

Supported Commands

Following is a list of commonly used SQL commands. They must be prefixed by SQL. See the *CA-Easytrieve Language Reference Guide* for a complete list. See your SQL vendor reference manuals for more information.

```
CLOSE COMMIT
CONNECT    DECLARE
DELETE    FETCH
INSERTOPEN
PUT    ROLLBACK
UPDATE
```

DB2

All DB2 commands that can be executed using the DYNAMIC execution facilities of DB2 are supported. See the *DATABASE 2 SQL Reference (SC26-4380)* manual for more information.

SQL/DS

All SQL/DS commands that are supported through the EXTENDED DYNAMIC facilities of SQL/DS are supported. See either the *SQL/Data System Application Programming for VSE (SH24-5018)* or the *SQL/Data System Application Programming for VM/SP (SH24-5068)* for more information.

CA-Datacom/PC

All CA-Datacom/PC SQL commands are supported. See the *CA-Datacom/PC SQL Programming and Reference Guide* for more information.

CA-Ingres

All CA-Ingres SQL commands that can be executed using the DYNAMIC execution facilities are supported. See the *CA-Ingres SQL Reference Guide* for information.

ORACLE

All ORACLE SQL commands that can be executed using the DYNAMIC execution facilities are supported. See the *ORACLE SQL Reference Guide* for information.

Note: Due to the ORACLE restriction against using the CURRENT OF clause with dynamic SQL, CA-Easytrieve must mimic the CURRENT OF clause using ROWID. This technique prohibits use of SELECT * in conjunction with DELETE or UPDATE WHERE CURRENT OF cursor.

Note: ORACLE systems are limited to a maximum of 10 cursors.

Unsupported SQL Commands

The following SQL commands cannot be issued using CA-Easytrieve controlled SQL processing:

```
BEGIN DECLARE      CREATE PROGRAM
DECLARE STATEMENT  DECLARE TABLE
DESCRIBE          END DECLARE
EXECUTE           EXECUTE IMMEDIATE
PREPARE           SELECT ... INTO ...
WHENEVER
```

The SELECT ...INTO... command is valid when processing DB2 static-only programs. Refer to the explanation for the SQLSYNTAX parameter for additional information.

Note: The SELECT ... INTO ... command (single SELECT) can be simulated using CA-Easytrieve automatic cursor management.

Retrieving All Columns

The following example retrieves all columns from the PERSONNEL table.

```

PARAM USERID('SQLDBA' 'SQLDBAPW') +
      PREPNAME(EASYOL 'SQLDBA')
DEFINE EMPNAME      W 20 A
DEFINE WORKDEPT     W  2 P 0
DEFINE EMPPHONE     W  3 P 0
DEFINE DEPTNAME     W 22 A      VARYING
DEFINE DEPTNUMBER   W  2 P 0
DEFINE NULLPHONE    W  2 B 0 . * NULL INDICATOR
DEFINE USERID       W  8 A      VALUE ('SQLDBA')
DEFINE PASSWORD     W  8 A      VALUE ('SQLDBAPW')
SQL DECLARE CURSOR1 CURSOR FOR      +
SELECT *              +
FROM PERSONNEL
JOB INPUT NULL NAME(SQLEX4)
SQL CONNECT :USERID IDENTIFIED BY :PASSWORD
PERFORM CHECKSQL
SQL OPEN CURSOR1
PERFORM CHECKSQL
DO WHILE SQLCODE NE 100. * 1403 FOR ORACLE
  SQL FETCH CURSOR1      +
    INTO :EMPNAME, :WORKDEPT, :EMPPHONE :NULLPHONE
  PERFORM CHECKSQL
  IF NULLPHONE < 0      . * PHONE PRESENT?
    EMPPHONE = 0      . * NO, SET TO 0
  END-IF
  IF SQLCODE NE 100    . * NOT END OF TABLE
    PRINT PERSNL
  END-IF
END-DO
SQL CLOSE CURSOR1
PERFORM CHECKSQL
STOP
CHECKSQL. PROC
  IF SQLCODE NE 0 AND SQLCODE NE 100. * 1403 FOR ORACLE
    DISPLAY 'SQLCODE = ' SQLCODE
  STOP EXECUTE
END-IF
END-PROC
REPORT PERSNL LINESIZE 65
      LINE EMPNAME WORKDEPT EMPPHONE

```

Reassign Departments

The next example reassigns all employees in department 901 to department 109 and displays the names of the employees. The PARM statement parameters are necessary only if you want to access a DB2 or CA-Ingres database subsystem other than the default.

```

PARAM SSID('DB2B')
DEFINE EMPNAME      W 20 A
DEFINE WORKDEPT     W  2 P 0
SQL DECLARE CURSOR1 CURSOR FOR      +
SELECT EMPNAME      +
FROM PERSONNEL      +
WHERE WORKDEPT = 901      +
FOR UPDATE OF WORKDEPT
JOB INPUT NULL NAME(SQLEX5)
SQL CONNECT :USERID IDENTIFIED BY :PASSWORD
PERFORM CHECKSQL
SQL OPEN CURSOR1
PERFORM CHECKSQL

```

```

DO WHILE SQLCODE NE 100. * 1403 FOR ORACLE
  SQL FETCH CURSOR1
    INTO :EMPNAME
  PERFORM CHECKSQL
  IF SQLCODE NE 100 . * 1403 FOR ORACLE
    PRINT PERSNL
    SQL UPDATE PERSONNEL
      SET WORKDEPT = 109
      WHERE CURRENT OF CURSOR1
    PERFORM CHECKSQL
  END-IF
END-DO
SQL CLOSE CURSOR1
PERFORM CHECKSQL
STOP
CHECKSQL. PROC
  IF SQLCODE NE 0 AND SQLCODE NE 100. * 1403 FOR ORACLE
    DISPLAY 'SQLCODE = ' SQLCODE
  STOP EXECUTE
END-IF
END-PROC
REPORT PERSNL LINESIZE 65
LINE EMPNAME

```

Update Phone Numbers

The next example illustrates how to update a phone system. In this case all phone numbers must be changed to 5 digits. The first character must be a 7 and the rest of the digits remain the same. If an employee does not have a phone number, his or her record is not updated. No update report is necessary.

```

JOB INPUT NULL NAME(SQLEX6)
SQL CONNECT :USERID IDENTIFIED BY :PASSWORD
PERFORM CHECKSQL
SQL UPDATE PERSONNEL
  SET EMPPHONE = 70000 + EMPPHONE
  WHERE EMPPHONE IS NOT NULL
PERFORM CHECKSQL
STOP
CHECKSQL. PROC
  IF SQLCODE NE 0 AND SQLCODE NE 100. * 1403 FOR ORACLE
    DISPLAY 'SQLCODE = ' SQLCODE
  STOP EXECUTE
END-IF
END-PROC

```

Using Table Name as Host Variable with Indicator Array

In this example, INDARRAY is used instead of the default indicator created by the SQL INCLUDE statement. An indicator is matched for each field in the PERSONNEL table.

```

PARM USERID('SQLDBA' 'SQLDBAPW') +
  PREPNAME(EASYOL 'SQLDBA')
SQL INCLUDE (EMPNAME, WORKDEPT, EMPPHONE) FROM PERSONNEL NULLABLE
DEFINE INDARRAY W 2 B 0 OCCURS 3
SQL DECLARE CURSOR1 CURSOR FOR
  SELECT * FROM PERSONNEL
JOB INPUT NULL NAME(SQLEXT)
  SQL CONNECT :USERID IDENTIFIED BY :PASSWORD
  PERFORM CHECKSQL
  SQL OPEN CURSOR1
  PERFORM CHECKSQL
  DO WHILE SQLCODE EQ 0
    SQL FETCH CURSOR1 +
      INTO :PERSONNEL :INDARRAY
    PERFORM CHECKSQL
    IF INDARRAY(3) < 0
      EMPPHONE = 0
    END-IF
    IF SQLCODE NE 100. * 1403 FOR ORACLE
      PRINT PERSNL
    END-IF
  END-DO
  SQL CLOSE CURSOR1
  PERFORM CHECKSQL
  STOP
CHECKSQL. PROC
  IF SQLCODE NE 0 AND SQLCODE NE 100. * 1403 FOR ORACLE
    DISPLAY 'SQLCODE = ' SQLCODE
  STOP EXECUTE
  END-IF
END-PROC
REPORT PERSNL LINESIZE 65
  LINE EMPNAME WORKDEPT EMPPHONE

```

Data Types Supported on the Workstation

The Workstation SQL interface supports both the EBCDIC and ASCII code systems; however, CA-Datacom/PC uses the ASCII code system. When you are processing EBCDIC data, additional processing time is required for conversion between code systems.

The Workstation SQL interface supports all NUMERIC data; however, CA-Datacom/PC uses the number "7" to denote a negative number. If either the EBCDIC code system or a sign digit other than "7" is used, a code conversion must be done before and after every database access.

The Workstation SQL interface supports both B or I binary types; however, CA-Datcom/PC uses only I. For performance reasons, you should specify an I data type for the SQL COMP field type (SQLCOMP) on the Datcom SQL Options panel in the Site Options Table. SQLCOMP specifies the internal data type used when including SQL catalog information using the SQL INCLUDE statement. See the *CA-Easytrieve/Workstation User Guide* for more information about the Site Options Table.

Introduction

CA-Easytrieve provides optional processing facilities that interface with CA-IDMS databases and with the CA-IDMS Integrated Data Dictionary (IDD).

CA-IDMS Interface

The CA-IDMS interface provides complete facilities for information retrieval from, and maintenance of, CA-IDMS databases. To use this interface effectively, you should have a basic knowledge of CA-IDMS and of the database(s) to be processed.

You can access an CA-IDMS database in one of two ways:

- Using automatic input
- Using controlled processing, which incorporates statements similar to those used in COBOL.

With automatic input (also called path processing), you can sweep an entire area of the database or retrieve records under the control of a tickler file or integrated indexing.

If the IDD interface is not used to generate definitions, the database administrator should build the FILE, RECORD, LOGICAL-RECORD, ELEMENT-RECORD, and DEFINE statements to describe the subschemas used. The subschema descriptions can then be stored in a CA-Easytrieve macro library.

IDD Interface

The IDD interface automatically generates definitions (taken from the IDD) for files, records, logical records, element records, and fields. This greatly reduces the effort often associated with database processing.

CA-IDMS Functionality

CA-Easytrieve's portability is constrained by CA-IDMS's portability. Where differences exist, CA-Easytrieve attempts to resolve the difference while still allowing your program to execute. Therefore, some parameters may be ignored where they do not function.

CA-Easytrieve CA-IDMS Statements

The following statements are used to define CA-IDMS database activities:

Statement(s)	Description
IDD statements	Retrieve definitions from the Integrated Data Dictionary
FILE	Identify a CA-IDMS database
RECORD	Describe database records
LOGICAL-RECORD	Describe logical records
ELEMENT-RECORD	Describe the database records that are part of a logical record
RETRIEVE	Describe automatic (path) processing for database records
SELECT	Describe automatic (path) processing for logical records
IDMS statements	Provide controlled retrieval and maintenance for both database and logical records

See the *CA-Easytrieve Language Reference Guide* for complete syntax for these statements.

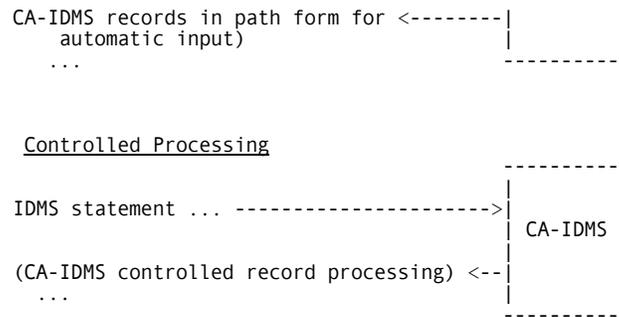
Processing Overview

The following exhibit gives an overview of how CA-Easytrieve interacts with a CA-IDMS database.

```
FILE file-name-1 IDMS
RECORD record-name
JOB INPUT (file-name-1) ...
```

Automatic Input

```
RETRIEVE file-name-1 ... -----> | CA-IDMS |
```



CA-IDMS Processing on the Workstation

CA-IDMS programs can be developed and targeted for execution on the workstation or they can be developed and tested, then ported to the mainframe. The following items are important when accessing CA-IDMS on the workstation.

Data Code System

The mainframe CA-IDMS data area is normally stored as EBCDIC and the workstation data area is stored in ASCII (there are exceptions). To process data in a CA-IDMS database, CA-Easytrieve must know the code system of the data. The code system controls how alphanumeric and zoned-numeric data are treated. The CODE ASCII parameter of the PARM statement is ignored on the mainframe, therefore, you can port your programs without changing platforms. You can either use the CODE parameter of the PARM statement (which becomes the default for the entire program) or the FILE statement coded for the CA-IDMS database.

For example:

```
PARM CODE PROCESS ASCII
```

Or:

```
FILE... CODE ASCII
```

Note: The PROCESS Site Option can be set to define the system-wide processing code system. See the Execution Options Panel in Chapter 2 of the *CA-Easytrieve/Workstation User Guide* for more information.

Note: If you use IDD statements to automatically generate FILE statements, the CODE parameter is not generated. You should use the PARM statement in this case.

Note: In UNIX, data is assumed to be ASCII.

Field Data Types

ASCII files can have fields defined for all standard data types, including alphanumeric, zoned-numeric, packed decimal, and binary. CA-Easytrieve stores and accesses data using fields you have defined. It is your responsibility to define the correct data types. This is critical when accessing data created by or accessed by other software products.

Zoned-numeric data in ASCII files can have various formats for negative data. You should use the ASCSIGN Site Option to specify the format when creating negative data. CA-Easytrieve automatically uses any of the supported formats for input. See the Execution Options Panel in Chapter 2 of the *CA-Easytrieve/Workstation User Guide* for more information.

Computational data (USAGE COMP) can be stored either as binary (B) fields, as on the mainframe, or as integer fields (I). CA-IDMS files normally use I fields for computational data. You can define your own I or B fields as needed. When fields are generated using the IDD interface, CA-Easytrieve uses the IDDCOMP Site Option to determine which type of field to generate. When IDDCOMP is set to B, computational fields are generated as mainframe binary (B) type fields. When IDDCOMP is set to I, computational fields are generated as integer (I) type fields. See the IDMS Options Panel in the *CA-Easytrieve/Workstation User Guide* for more information.

When porting your program, it is important to consider the definitions of computational data. Mainframe CA-Easytrieve programs may contain binary field definitions. Test data on the workstation may contain integer data. You must then change B data type fields to I data type fields or use IDD statements to automatically define your fields. This allows your program to remain fully portable.

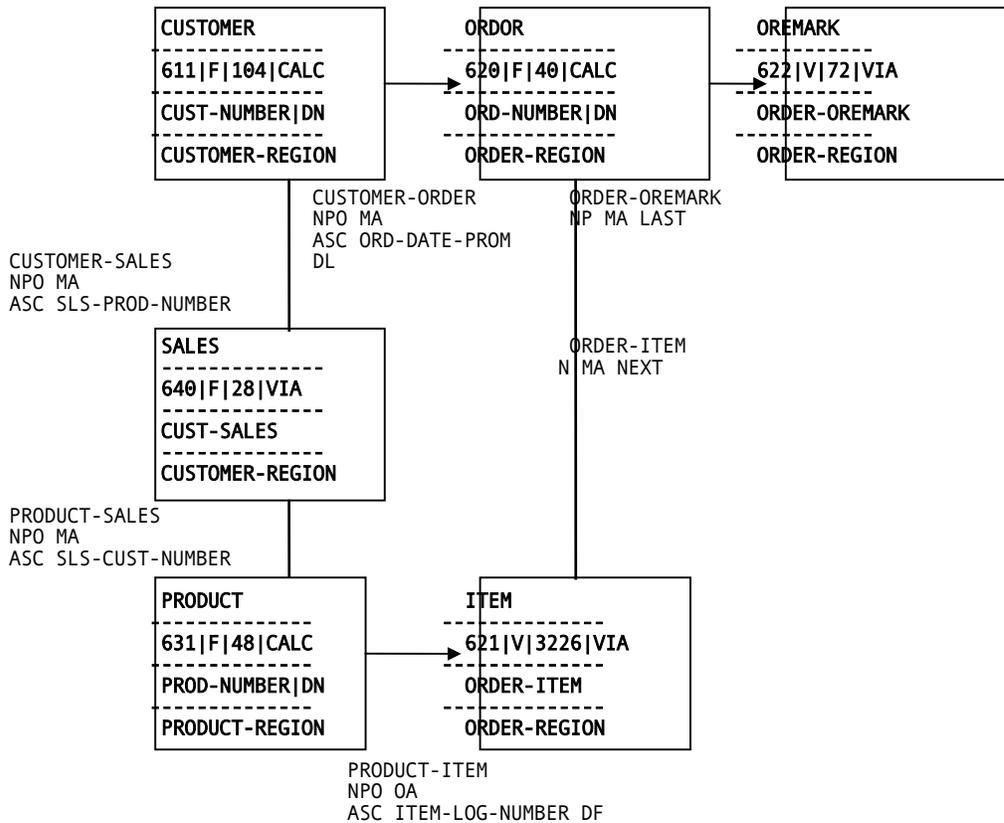
CA-Easytrieve automatically converts binary data to and from integer data for statements that require binary fields as parameters. Similarly, binary fields in the CA-IDMS Communications Block (IDMSCOM) are automatically converted as needed.

CA-IDMS Entity Names

CA-IDMS entity names such as records, sets, or areas, that are kept in the program source are already in ASCII on the workstation. When supplied as dynamic parameters, CA-Easytrieve converts them to ASCII as needed.

Sample CA-IDMS Database

The following exhibit illustrates a portion of the database used in examples in this guide. The CA-Easytrieve field definitions for this portion follow the exhibit.



Field Definitions

The following field definitions describe the sample database shown in the previous exhibit. These definitions are established by the database administrator and stored in a CA-Easytrieve macro.

```
FILE DBASE IDMS(DEMOSS03)
RECORD CUSTOMER 104 KEY(CUST-NO)
  CUST-NO 1 10 A
  CUST-NAME 11 20 A
RECORD ORDOR 40
  ORD-NO 1 7 A
  ORD-CPO# 8 10 A
RECORD OREMARK 72
  ORD-SEQ 1 2 A
  ORD-TEXT 3 70 A
RECORD SALES 28
  SLS-CUST-NO 1 10 A
RECORD PRODUCT 48
  PROD-NO 1 8 A
  PROD-DESC 9 20 A
RECORD ITEM 3226
```

ITEM-PROD#

1 8 A

Sample Logical Record

A description of the logical record definition for our database example as defined in CA-IDMS is as follows:

```

ADD LOGICAL RECORD NAME IS CUST-SALES-LR
ELEMENTS ARE CUSTOMER
                SALES
                PRODUCT

COMMENTS .
'*****'
- '
- 'LR VERBS ALLOWED: ALL
- '
- '*****'

ADD PATH-GROUP OBTAIN CUST-SALES-LR

SELECT FOR FIELDNAME-EQ CUST-NUMBER
OBTAIN EMPLOYEE WHERE CALCKEY EQ CUST-NUMBER
OF REQUEST
ON 0326 CLEAR RETURN LR-NOT-FOUND
OBTAIN EACH SALES WITHIN CUSTOMER-SALES
OBTAIN EACH PRODUCT WITHIN PRODUCT-SALES

SELECT FOR FIELDNAME-EQ PROD-NUMBER
OBTAIN PRODUCT WHERE CALCKEY EQ PROD-NUMBER
OF REQUEST
ON 0326 CLEAR RETURN LR-NOT-FOUND
OBTAIN OWNER WITHIN PRODUCT-SALES
OBTAIN OWNER WITHIN CUSTOMER-SALES

SELECT
OBTAIN EACH CUSTOMER WITHIN CUSTOMER-REGION
OBTAIN EACH SALES WITHIN CUSTOMER-SALES
OBTAIN EACH PRODUCT WITHIN PRODUCT-REGION

ADD PATH-GROUP MODIFY CUST-SALES-LR

SELECT
MODIFY SALES.

ADD PATH-GROUP ERASE CUST-SALES-LR

SELECT
ERASE SALES.

ADD PATH-GROUP STORE CUST-SALES-LR

SELECT
STORE SALES.

```

Logical Record Definition

The CA-Easytrieve field definitions that follow describe the sample logical record shown above. These definitions are established by the database administrator and stored in a CA-Easytrieve macro.

```

*
  IDD NAME DBNAME 'TSTDICT'
*
  FILE DEMOSSLR IDMS (DEMOSSLR)
    LOGICAL-RECORD CUST-SALES-LR

```

```

ELEMENT-RECORD CUSTOMER
  CUST-NUMBER      1  10  A
  CUST-NAME       11  20  A
ELEMENT-RECORD SALES
  SLS-CUST-NO     1  10  A
ELEMENT-RECORD PRODUCT
  PROD-NUMBER     1   8  A
  PROD-DESC       9  20  A
*
JOB INPUT NULL
STOP

```

IDD Interface

The interface between CA-Easytrieve and the CA-IDMS Integrated Data Dictionary (IDD) is accomplished by the use of IDD statements. To give you the fullest control over this access, CA-Easytrieve provides the following IDD statements:

Statement	Description
IDD NAME	Control which dictionary is accessed.
IDD VERSION	Specify which version of the information is to be retrieved.
IDD SUBSCHEMA	Retrieve definitions of CA-IDMS files (subschemas).
IDD FILE	Retrieve definitions of non-CA-IDMS files.
IDD RECORD	Retrieve definitions of records.

IDD statements generate CA-Easytrieve library definitions based on the parameters coded on the statement. IDD statement parameters direct the retrieval of data definition information from the IDD and insert definitions for files, database records, logical records, element records, and fields directly into CA-Easytrieve internal tables.

The actual FILE, RECORD, LOGICAL-RECORD, ELEMENT-RECORD, and DEFINE statements are not generated as visible source code, and will not appear in the output listing. Instead, the IDD NAME statement simulates the presence of these statements by storing the information obtained from the IDD in the same manner that the statements would. The DMAP parameter of the PARM statement can be used to obtain a description of the information stored.

Note: IDD statements are compatible with operating systems using CA-IDMS/PC release 2.50 or above.

See the *CA-Easytrieve Language Reference Guide* for complete syntax for the above statements.

Program Name

If a subschema has restricted authorization, a properly-registered program name must be supplied to CA-Easytrieve. This program name can be specified on the IDD NAME statement. If the IDD NAME statement is not coded, CA-Easytrieve uses a default name of EASYPLUS. The default program name would have to have been previously stored in the IDD definition for the restricted subschema.

Conforming IDD Item Descriptions to CA-Easytrieve Standards

To ensure the correct translation of certain IDD record constructs into CA-Easytrieve formats, the following adaptations have been made:

- For an item that occurs multiple times in a record and is supplied with an index, the index name is used as the CA-Easytrieve INDEX name. If the index is not supplied in the IDD, a CA-Easytrieve INDEX name is generated by concatenating the item name with the string +INDEX.
- If the item being defined is part of a group item, the relationship between the defined item and its containing group item is preserved by CA-Easytrieve. The group item is defined as a segmented data item (see Table Processing and Array Processing in the “Coding a CA-Easytrieve Program” chapter for a discussion of segmented data). If the group item occurs multiple times, then the defined item also occurs multiple times.
- Data types that do not conform to the rules of CA-Easytrieve are treated as alphanumeric (A) fields.

Handling of Group Item Definition

When the IDD interface creates a definition of a group item and its component items, it is possible that an ambiguity might occur, making the definition unusable by your CA-Easytrieve program. This ambiguity is created when a component of the group item and another item within the same database record have identical names. The component item name, either alone or with the record name as a qualifier, is not sufficient to distinguish between the two items. To resolve this ambiguity, CA-Easytrieve enables you to use the group item’s name as a qualifier for any item it contains.

Since the group item itself might be part of a larger group item, it is possible for an item defined with the IDD interface to use many levels of qualification. The only limit on the number of levels is the number of levels the IDD allows you to define for the record definition in the dictionary.

Note: On the workstation, the MAXQUAL Site Option controls the maximum number of levels allowed in CA-Easytrieve. The minimum setting when using CA-IDMS is 4. However, it is recommended that you set MAXQUAL to a minimum of 8. If you define a level that exceeds the MAXQUAL setting, you receive an error. If you set MAXQUAL too high, you use excessive memory. See the Compiler Options Panel in the *CA-Easytrieve/Workstation User Guide* for more information.

As a result, the syntax of a reference to an item defined with the IDD interface is:

```
[file-name :] [record-name :] [group-item-name : ...] field-name
```

for an item defined in a database record and:

```
[file-name :] [logical-record-name :] [element-record-name :] +
[group-item-name : ...] field-name
```

for an item defined in a logical record.

Examples

Defining a Subschema

```
PARM DEBUG (DMAP)
*
IDD SUBSCHEMA DEMOSS03 SCHEMA DEMOSCHM
*
JOB INPUT NULL
STOP
```

Defining a Logical Record of a Subschema

```
PARM DEBUG (DMAP)
*
IDD SUBSCHEMA DEMOSSLR SCHEMA DEMOSCHM +
  SELECT (CUST-SALES-LR)
*
JOB INPUT NULL
STOP
```

Defining Only Select Records from a Subschema

```
PARM DEBUG (DMAP)
*
IDD VERSION SCHEMA 100
IDD SUBSCHEMA DEMOSS03 SCHEMA DEMOSCHM +
  SELECT (SALES OREMARK)
*
JOB INPUT NULL
STOP
```

IDMS Interface

When using the CA-IDMS interface, you can define subschemas using the following statements:

Statement	Description
FILE	Identify the database to be processed.
RECORD	Identify the database records available for automatic or controlled processing.
LOGICAL-RECORD	Identify the logical records available for automatic or controlled processing.
ELEMENT-RECORD	Identify the element records that comprise the logical record.

Examples of these statements were shown previously (See Logical Record Definition) and can be used by the database administrator to establish database field definitions. These definitions can then be stored in a CA-Easytrieve macros for your access. You can also generate these statements automatically using the IDD interface.

See the *CA-Easytrieve Language Reference Guide* for complete syntax for the above statements.

Note: Logical records are supported on the workstation.

Communications Block

When the first IDMS FILE statement is encountered, a CA-IDMS Communications Block is created in "S" working storage. The fields generated in CA-Easytrieve (mainframe and workstation), and in CA-Easytrieve UNIX, are as follows:

CA-Easytrieve IDMS Communications Block

```

DEFINE IDMSCOM      S          216  A
DEFINE IDMSNAME     IDMSCOM    8  A, VALUE 'EASYPLUS'
DEFINE IDMSSTATUS   IDMSCOM + 8  4  A
DEFINE IDMSKEY      IDMSCOM + 12 4  B 0, MASK HEX
DEFINE IDMSREC      IDMSCOM + 16 16  A
DEFINE IDMSNODE     IDMSCOM + 16 8  A
DEFINE IDMSDB       IDMSCOM + 24 8  A
DEFINE IDMSAREA     IDMSCOM + 32 16  A
DEFINE IDMSDICTNODE IDMSCOM + 32 8  A
DEFINE IDMSDICTNAME IDMSCOM + 40 8  A
DEFINE IDMSESET     IDMSCOM + 48 16  A
DEFINE IDMSEREC     IDMSCOM + 64 16  A
DEFINE IDMSEAREA    IDMSCOM + 80 16  A
DEFINE IDMSCON      IDMSCOM + 96 1  A, OCCURS 100, INDEX IDMSCON-INDEX
DEFINE IDMSCON02    IDMSCOM + 97 1  A. * FINISH
DEFINE IDMSCON03    IDMSCOM + 98 1  A. * ERASE PERMANENT
DEFINE IDMSCON04    IDMSCOM + 99 1  A. * ERASE ALL
DEFINE IDMSCON06    IDMSCOM +101 1  A. * FIND DB-KEY REC
DEFINE IDMSCON07    IDMSCOM +102 1  A. * FIND CURRENT REC
DEFINE IDMSCON08    IDMSCOM +103 1  A. * FIND CURRENT SET
DEFINE IDMSCON09    IDMSCOM +104 1  A. * FIND CURRENT AREA
DEFINE IDMSCON10    IDMSCOM +105 1  A. * FIND NEXT REC SET
DEFINE IDMSCON11    IDMSCOM +106 1  A. * FIND NEXT REC AREA
DEFINE IDMSCON12    IDMSCOM +107 1  A. * FIND PRIOR REC SET

```

DEFINE	IDMSCON13	IDMSCOM +108	1	A.	*	FIND PRIOR REC AREA
DEFINE	IDMSCON14	IDMSCOM +109	1	A.	*	FIND NEXT SET
DEFINE	IDMSCON15	IDMSCOM +110	1	A.	*	FIND NEXT AREA
DEFINE	IDMSCON16	IDMSCOM +111	1	A.	*	FIND PRIOR SET
DEFINE	IDMSCON17	IDMSCOM +112	1	A.	*	FIND PRIOR AREA
DEFINE	IDMSCON18	IDMSCOM +113	1	A.	*	FIND FIRST REC SET
DEFINE	IDMSCON19	IDMSCOM +114	1	A.	*	FIND FIRST REC AREA
DEFINE	IDMSCON20	IDMSCOM +115	1	A.	*	FIND FIRST SET
DEFINE	IDMSCON21	IDMSCOM +116	1	A.	*	FIND FIRST AREA
DEFINE	IDMSCON22	IDMSCOM +117	1	A.	*	FIND LAST REC SET
DEFINE	IDMSCON23	IDMSCOM +118	1	A.	*	FIND LAST REC AREA
DEFINE	IDMSCON24	IDMSCOM +119	1	A.	*	FIND LAST SET
DEFINE	IDMSCON25	IDMSCOM +120	1	A.	*	FIND LAST AREA
DEFINE	IDMSCON30	IDMSCOM +125	1	A.	*	FIND CURRENT
DEFINE	IDMSCON31	IDMSCOM +126	1	A.	*	FIND OWNER SET
DEFINE	IDMSCON32	IDMSCOM +127	1	A.	*	FIND CALC
DEFINE	IDMSCON33	IDMSCOM +128	1	A.	*	FIND REC SET USING
DEFINE	IDMSCON34	IDMSCOM +129	1	A.	*	GET REC
DEFINE	IDMSCON35	IDMSCOM +130	1	A.	*	MODIFY
DEFINE	IDMSCON36	IDMSCOM +131	1	A.	*	READY UPDATE
DEFINE	IDMSCON37	IDMSCOM +132	1	A.	*	READY RETRIEVAL
DEFINE	IDMSCON38	IDMSCOM +133	1	A.	*	READY UPDATE PROT
DEFINE	IDMSCON39	IDMSCOM +134	1	A.	*	READY RETRIEVE PROT
DEFINE	IDMSCON40	IDMSCOM +135	1	A.	*	READY RETRIEVE EXCL
DEFINE	IDMSCON41	IDMSCOM +136	1	A.	*	READY UPDATE EXCL
DEFINE	IDMSCON42	IDMSCOM +137	1	A.	*	STORE
DEFINE	IDMSCON43	IDMSCOM +138	1	A.	*	GET
DEFINE	IDMSCON44	IDMSCOM +139	1	A.	*	CONNECT
DEFINE	IDMSCON46	IDMSCOM +141	1	A.	*	DISCONNECT
DEFINE	IDMSCON48	IDMSCOM +143	1	A.	*	BIND REC
DEFINE	IDMSCON50	IDMSCOM +145	1	A.	*	FIND DUP
DEFINE	IDMSCON51	IDMSCOM +146	1	A.	*	FIND REC SET CURR USING
DEFINE	IDMSCON52	IDMSCOM +147	1	A.	*	ERASE MEMBER
DEFINE	IDMSCON53	IDMSCOM +148	1	A.	*	ERASE SELECTIVE
DEFINE	IDMSCON54	IDMSCOM +149	1	A.	*	ACCEPT
DEFINE	IDMSCON55	IDMSCOM +150	1	A.	*	ACCEPT RECORD
DEFINE	IDMSCON56	IDMSCOM +151	1	A.	*	ACCEPT AREA
DEFINE	IDMSCON57	IDMSCOM +152	1	A.	*	ACCEPT SET
DEFINE	IDMSCON59	IDMSCOM +154	1	A.	*	BIND SUBSCHEMA
DEFINE	IDMSCON60	IDMSCOM +155	1	A.	*	IF MEMBER
DEFINE	IDMSCON62	IDMSCOM +157	1	A.	*	IF NOMEMBER
DEFINE	IDMSCON64	IDMSCOM +159	1	A.	*	IF EMPTY
DEFINE	IDMSCON65	IDMSCOM +160	1	A.	*	IF NOEMPTY
DEFINE	IDMSCON66	IDMSCOM +161	1	A.	*	COMMIT
DEFINE	IDMSCON67	IDMSCOM +162	1	A.	*	ROLLBACK
DEFINE	IDMSCON68	IDMSCOM +163	1	A.	*	ACCEPT NEXT SET
DEFINE	IDMSCON69	IDMSCOM +164	1	A.	*	ACCEPT PRIOR SET
DEFINE	IDMSCON70	IDMSCOM +165	1	A.	*	ACCEPT OWNER SET
DEFINE	IDMSCON71	IDMSCOM +166	1	A.	*	ACCEPT STATISTICS
DEFINE	IDMSCON73	IDMSCOM +168	1	A.	*	BIND PROCEDURE
DEFINE	IDMSCON74	IDMSCOM +169	1	A.	*	ACCEPT PROCEDURE
DEFINE	IDMSCON75	IDMSCOM +170	1	A.	*	FIND DB-KEY
DEFINE	IDMSCON76	IDMSCOM +171	1	A.	*	FIND NTH REC SET
DEFINE	IDMSCON77	IDMSCOM +172	1	A.	*	FIND NTH REC AREA
DEFINE	IDMSCON78	IDMSCOM +173	1	A.	*	FIND NTH SET
DEFINE	IDMSCON79	IDMSCOM +174	1	A.	*	FIND NTH AREA
DEFINE	IDMSCON81	IDMSCOM +176	1	A.	*	RETURN
DEFINE	IDMSCON82	IDMSCOM +177	1	A.	*	RETURN FIRST
DEFINE	IDMSCON83	IDMSCOM +178	1	A.	*	RETURN LAST
DEFINE	IDMSCON84	IDMSCOM +179	1	A.	*	RETURN NEXT
DEFINE	IDMSCON85	IDMSCOM +180	1	A.	*	RETURN PRIOR
DEFINE	IDMSCON86	IDMSCOM +181	1	A.	*	RETURN USING
DEFINE	IDMSCON87	IDMSCOM +182	1	A.	*	KEEP
DEFINE	IDMSCON88	IDMSCOM +183	1	A.	*	KEEP EXCLUSIVE
DEFINE	IDMSCON89	IDMSCOM +184	1	A.	*	KEEP REC
DEFINE	IDMSCON90	IDMSCOM +185	1	A.	*	KEEP EXCLUSIVE REC
DEFINE	IDMSCON91	IDMSCOM +186	1	A.	*	KEEP SET
DEFINE	IDMSCON92	IDMSCOM +187	1	A.	*	KEEP EXCLUSIVE SET
DEFINE	IDMSCON93	IDMSCOM +188	1	A.	*	KEEP AREA
DEFINE	IDMSCON94	IDMSCOM +189	1	A.	*	KEEP EXCLUSIVE AREA
DEFINE	IDMSCON95	IDMSCOM +190	1	A.	*	COMMIT ALL
DEFINE	IDMSCON96	IDMSCOM +191	1	A.	*	ROLLBACK CONTINUE
DEFINE	IDMSCON99	IDMSCOM +194	1	A.	*	LRF FUNCTION
DEFINE	IDMSDIRECT	IDMSCOM +196	4	B	0,	MASK HEX
DEFINE	IDMSRESV	IDMSCOM +200	7	N,		MASK HEX

```
DEFINE IDMSFILL IDMSCOM +207 1 N, MASK HEX  
DEFINE IDMSOCCUR IDMSCOM +208 4 B 0, MASK HEX  
DEFINE IDMSSEQ IDMSCOM +212 4 B 0, MASK HEX
```

CA-Easytrieve UNIX IDMS Communications Block

```

DEFINE IDMSCOM          S           216  A
DEFINE IDMSNAME         IDMSCOM      8  A, VALUE 'EASYPLUS'
DEFINE IDMSSTATUS       IDMSCOM + 8  4  A
DEFINE IDMSKEY          IDMSCOM + 12  4  B 0, MASK HEX
DEFINE IDMSREC          IDMSCOM + 16 16  A
DEFINE IDMSNODE        IDMSCOM + 16  8  A
DEFINE IDMSDB          IDMSCOM + 24  8  A
DEFINE IDMSAREA        IDMSCOM + 32 16  A
DEFINE IDMSDICTNODE    IDMSCOM + 32  8  A
DEFINE IDMSDICTNAME    IDMSCOM + 40  8  A
DEFINE IDMSESET        IDMSCOM + 48 16  A
DEFINE IDMSEREC        IDMSCOM + 64 16  A
DEFINE IDMSEAREA       IDMSCOM + 80 16  A
DEFINE IDMSCON         IDMSCOM + 96  1  A, OCCURS 100
DEFINE IDMSDIRECT      IDMSCOM +196  4  B 0, MASK HEX
DEFINE IDMSRESV        IDMSCOM +200  7  N, MASK HEX
DEFINE IDMSFILL        IDMSCOM +207  1  N, MASK HEX
DEFINE IDMSOCCUR       IDMSCOM +208  4  B 0, MASK HEX
DEFINE IDMSSEQ         IDMSCOM +212  4  B 0, MASK HEX

```

Logical Record Communications Block

When the first Logical Record statement is encountered, a Logical Record Communications Block is created in "S" working storage. The fields generated are:

```

DEFINE SLC              S           1024  A
DEFINE SUBSCHEMA-LR-CTRL SLC        1024  A
DEFINE LRC-LRPXELNG    SLC           2  B
DEFINE LRC-MAXVXP      SLC +0002     2  B
DEFINE LRIDENT         SLC +0004     4  A
DEFINE LRVERB          SLC +0008     8  A
DEFINE LRNAME          SLC +0016    16  A
DEFINE LR-STATUS       SLC +0032    16  A
DEFINE LR-FILLER-01    SLC +0048    16  A
DEFINE LRPXE           SLC +0064     1  A +
                        OCCURS 960 INDEX LR-PXE-NDX
DEFINE PXE             LRPXE        256  A
DEFINE PXENEXT         LRPXE         4  B
DEFINE PXETABO         LRPXE +0004     2  B
DEFINE PXEDSPL         LRPXE +0006     2  B
DEFINE PXEDYN          LRPXE +0008     2  B
DEFINE PXEDLEN         LRPXE +0010     2  B
DEFINE PXENDEC         LRPXE +0012     1  A
DEFINE PXEDTYP         LRPXE +0013     1  A
DEFINE PXEOTYP         LRPXE +0014     1  A
DEFINE PXEFLAG         LRPXE +0015     1  A
DEFINE PXE-FILLER-01   LRPXE +0016    240  A

```

Using Logical and Element Records in Non-CA-IDMS Statements

Non-CA-IDMS statements in CA-Easytrieve treat database records in much the same way as files. That is, the database record can be written to a file using the FROM parameter of the PUT or WRITE statement, the contents of the record buffer can be accessed using the MOVE statement, and the fields of the record can be moved selectively using the MOVE LIKE statement. In addition, the definitions of all fields defined in the record can be copied to another record or file using the COPY statement.

For those non-CA-IDMS statements that allow record names to be used, a logical record is treated in exactly the same manner as a database record.

However, element records are not treated like database records. In non-CA-IDMS statements, element records are treated as alphanumeric fields. This means that you can use an element record name in any context where you can use an alphanumeric field.

Automatic Input

Automatic input is a facility whereby CA-Easytrieve retrieves information from the database and makes it available to the program. See *Controlled vs. Automatic Processing* in the “File Processing” chapter for a description of automatic input as it applies to conventional files. To indicate that automatic input is to occur for a database, you must do the following:

1. Code the INPUT parameter on the JOB statement for each activity that will require automatic input from the database. The INPUT parameter must specify the filename from the FILE statement that defines the subschema. This must be the only filename specified. CA-Easytrieve does not support synchronized file processing for CA-IDMS database files.
2. Code either a RETRIEVE statement or a SELECT statement following the JOB statement. Use a RETRIEVE statement to retrieve database records. Use a SELECT statement for logical records. Only one statement, RETRIEVE or SELECT, must be coded.

The RETRIEVE and SELECT statements describe the particular portion of the database to be retrieved. CA-Easytrieve performs all the calls needed to retrieve the information described by the automatic input statements. The sequence of processing for automatic input from a CA-IDMS database is shown in the following exhibit.

```

IF first call
  IDMS BIND subschema-name from INPUT file +
    PROGRAM-NAME parameter from RETRIEVE/SELECT +
    DBNAME parameter from RETRIEVE/SELECT +
    NODE parameter from RETRIEVE/SELECT +
    DICTNAME parameter from RETRIEVE/SELECT +
    DICTNODE parameter from RETRIEVE/SELECT
  IDMS BIND FILE file-name RECORD record-name (RETRIEVE only)
  ... (repeated for each record specified)
  IDMS READY ALL RETRIEVAL

END-IF
retrieve next set of information
IF no more information
  wrap up reports
  STOP
END-IF
... (your program processes the information)
GO TO JOB

```

The RETRIEVE statement provides for automatic input of CA-IDMS databases. Input is accomplished by either sweeping an entire database area and sequentially processing all occurrences of the root record, or by selectively processing root records through the use of a tickler file or integrated index.

Sweep of an Area

Sweeping an entire database area for all occurrences of the root record provides the default input. OBTAIN NEXT RECORD WITHIN AREA calls are issued at the root level until the database area has been exhausted. If specified, the INDEX, LIMIT, and WHILE subparameters control the sweep.

Tickler File Control

Optionally, a file of root record keys can control the extent of the database to be processed. The keys are obtained one-at-a-time from the tickler file. OBTAIN CALC calls are issued for each key in the tickler file. Only CALC records can be the root when the tickler file is used. The record must have the KEY parameter specified on the RECORD statement.

Input Definition (Paths)

Automatic input of CA-IDMS databases depends on the concept of path processing. Each database path, identified by the SELECT parameter, is processed in a top-to-bottom order. A root record is obtained first, then path access continues downward through the records coded in the SELECT parameter. When the end of each path is reached, that data is made available to the program as an input record.

If another path is defined, denoted by a repeated record name or node, that path is then processed until end of path. When all paths for the root have been exhausted, the next root is obtained. Paths can be defined from a member occurrence to its owner occurrence if owner pointers exist for the set.

Records at each level of the path below the root are retrieved using OBTAIN NEXT RECORD WITHIN SET if the owner record type is at the higher level. If the member record type is specified at the higher level, then OBTAIN OWNER calls are used. The name of the set to be used must be specified with the SET subparameter of the SELECT parameter entry for the lower level record.

Automatic Input of Logical Records

The SELECT statement provides for automatic input of logical records from CA-IDMS databases. The input is a sequential retrieval of all occurrences of a specified logical record that satisfy a user-specified WHERE clause. OBTAIN NEXT RECORD WHERE calls are issued for the logical record until all records have been retrieved. A logical path is not input if LR-STATUS is LR-ERROR or LR-NOT-FOUND.

WHERE Clause

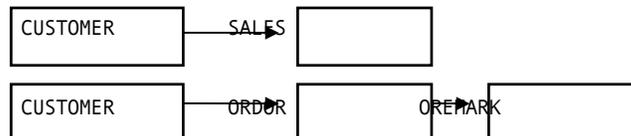
The Boolean expression required by the WHERE parameter of the SELECT statement is coded using the syntax required by CA-IDMS for COBOL programs. The only difference is that if the Boolean expression extends over multiple source records, each record must be continued using the CA-Easytrieve conventions described in the "Overview" chapter of the *CA-Easytrieve Language Reference Guide*.

See the *Programmer's Reference Guide - COBOL* for a description of the syntax of a Boolean expression.

Examples

Processing Two Distinct Paths from a Single Root

This example illustrates path processing. The RETRIEVE statement returns all data to the program for processing. Information about missing data is also available.



```

FILE DBASE IDMS(DEMOSS03)
RECORD CUSTOMER 104
  CUST-NO      1 10 A
  CUST-NAME   11 20 A
RECORD ORDOR  40
  ORD-NO      1  7 A
  ORD-CPO#    8 10 A
RECORD OREMARK 72
  ORD-SEQ     1  2 A
  ORD-TEXT    3 70 A
RECORD SALES   28
  SLS-CUST-NO 1 10 A
JOB INPUT (DBASE) NAME TWO-DISTINCT-PATHS
RETRIEVE DBASE +
  SELECT (CUSTOMER AREA 'CUSTOMER-REGION' +
    SALES ID 'SA' SET 'CUSTOMER-SALES' +
    CUSTOMER +
    ORDOR SET 'CUSTOMER-ORDER' +
    OREMARK ID 'RE' SET 'ORDER-OREMARK')
  IF PATH-ID EQ 'RE'
  
```

```
        DISPLAY ORD-TEXT  
    ELSE  
        DISPLAY SLS-CUST-NO  
    END-IF
```

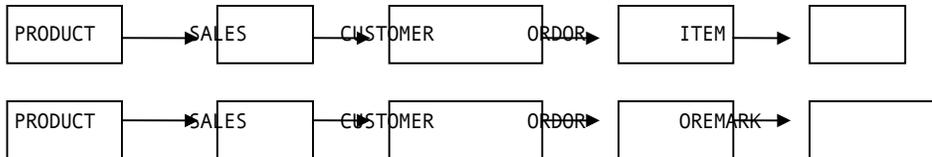
The first customer record in the area is obtained. The first sales record for the customer is then obtained. If no sales exist, the order path is processed. If a sale record exists for the customer, the path containing the customer and sales record is returned to the program. This processing continues until no more sales records exist for the customer. The order path is then processed.

The first order for the root customer is then obtained, as well as the first remark for the order. This path is then returned to the program. Next, the second remark for the first order is obtained and the path is returned. The customer and order records remain unchanged. Only the remark record is affected. When all remarks for the first order are returned, the next order for the customer is obtained with all its remarks. This processing continues until all orders and all remarks are obtained and returned to the program. The second customer root is then obtained and processing continues as described above until all customer records have been processed.

PATH-ID is used to test which path is returned to the program. When PATH-ID = SA, the customer sales path is available. When PATH-ID = RE, the customer/order/remark path is available. Paths for customers without sales or without orders are not processed by this program.

Processing Two Paths with Intermediate Records the Same

Like the previous example, the program in this example also processes two paths. The product, sales, customers, and order records are all processed. Then the first path specifies that the item records for each order are to be returned and then the remark records for the same order are returned. Each record also has an ID specified.



```

FILE DBASE IDMS(DEMOSS03)
RECORD CUSTOMER 104
  CUST-NO      1 10 A
  CUST-NAME   11 20 A
RECORD ORDOR  40
  ORD-NO      1  7 A
  ORD-CPO#    8 10 A
RECORD OREMARK 72
  ORD-SEQ     1  2 A
  ORD-TEXT    3 70 A
RECORD SALES   28
  SLS-CUST-NO 1 10 A
RECORD PRODUCT 48
  PROD-NO     1  8 A
  PROD-DESC   9 20 A
RECORD ITEM   3226
  ITEM-PROD#  1  8 A
JOB INPUT (DBASE) NAME TWO-PATHS
RETRIEVE DBASE +
  SELECT (PRODUCT ID 'PR'   AREA 'PRODUCT-REGION' +
         SALES ID 'SA'     SET 'PRODUCT-SALES' +
         CUSTOMER ID 'CU'   SET 'CUSTOMER-SALES' +
         ORDOR ID 'OR'     SET 'CUSTOMER-ORDER' +
         ITEM ID 'IT'      SET 'ORDER-ITEM' +
         ORDOR +
         OREMARK ID 'RE'   SET 'ORDER-OREMARK')
IF PATH-ID EQ 'RE'
  DISPLAY PROD-DESC CUST-NAME ORD-TEXT
END-IF
IF PATH-ID EQ 'IT'
  DISPLAY PROD-DESC CUST-NAME ITEM-PROD#
END-IF
  
```

In a typical CA-IDMS database path, each record can occur multiple times or may not occur at all. Occasionally, you might want to determine not only which path is available but also which records in the path are available. Based on the previous example, the information in the following table is provided.

PATH-ID	Available RECORD(s)
PR	PRODUCT
SA	PRODUCT - SALES
CU	PRODUCT - SALES - CUSTOMER

OR	PRODUCT - SALES - CUSTOMER - ORDOR
IT	PRODUCT - SALES - CUSTOMER - ORDOR - ITEM
RE	PRODUCT - SALES - CUSTOMER - ORDOR - OREMARK

Tickler File Control of Root Records

The program in this example illustrates path processing using a tickler file to identify the root records to be processed.

```

FILE DBASE IDMS(DEMOSS03)
RECORD CUSTOMER 104 KEY(CUST-NO)
  CUST-NO      1 10 A
  CUST-NAME    11 20 A
FILE KEYS
KEY 1 10 N
JOB INPUT (DBASE) NAME TICKLER-FILE-CONTROL
  RETRIEVE DBASE +
  KEYFILE KEYS KEYVALUE(CUST-NO = KEY) +
  SELECT (CUSTOMER AREA 'CUSTOMER-REGION')
  IF PATH-ID EQ 'NF'
    DISPLAY 'ROOT RECORD NOT FOUND FOR ' KEY
  GO TO JOB
  END-IF
  DISPLAY CUST-NAME

```

Complete Path Processing

The program in this example illustrates how to bypass certain paths, specifically those paths whose lowest record is not present. This example selects only complete paths for data processing.

```

FILE DBASE IDMS(DEMOSS03)
RECORD CUSTOMER 104
  CUST-NO      1 10 A
  CUST-NAME    11 20 A
RECORD ORDOR 40
  ORD-NO      1 7 A
  ORD-CPO#    8 10 A
RECORD ITEM 3226
  ITEM-PROD#  1 8 A
JOB INPUT (DBASE) NAME COMPLETE-PATH-PROCESSING
  RETRIEVE DBASE +
  SELECT (CUSTOMER AREA 'CUSTOMER-REGION' +
  ORDOR SET 'CUSTOMER-ORDER' +
  ITEM ID 'IT' SET 'ORDER-ITEM')
  IF PATH-ID NE 'IT'
    GO TO JOB
  END-IF
  DISPLAY CUST-NAME ITEM-PROD#

```

Limited Record Retrieval

You can use the LIMIT subparameter of the SELECT statement to limit record occurrence retrieval. This example describes an area sweep where the number of root records retrieved is limited to five for program testing purposes. When the limit is reached, input processing is terminated.

```

FILE DBASE IDMS(DEMOSS03)
RECORD SALES 28
  SLS-CUST-NO 1 10 A
JOB INPUT (DBASE) NAME RETRIEVAL-LIMIT
  RETRIEVE DBASE +

```

```
SELECT (SALES AREA 'CUSTOMER-REGION' LIMIT 5)
DISPLAY SLS-CUST-NO
```

Another reason to limit record retrieval is to inhibit potential redundant calls to CA-IDMS. For instance, if it is known that a particular record never occurs more than twice in a path, code LIMIT 2 for that record. This use of LIMIT improves throughput for database activities.

Conditional Record Retrieval

You can screen any record to establish the acceptability of the record. CA-Easytrieve bypasses record occurrences that fail the acceptance test for input consideration. Use the WHILE condition to control record acceptance.

```
FILE DBASE IDMS (DEMOSS03)
RECORD CUSTOMER 104
  CUST-NO      1 10 A
  CUST-NAME    11 20 A
JOB INPUT (DBASE) NAME RECORD-PROCESSING
RETRIEVE DBASE +
  SELECT (CUSTOMER AREA 'CUSTOMER-REGION' +
         WHILE (CUST-NAME EQ 'JONES'))
DISPLAY CUST-NO
```

Select Statement

The following shows a logical record Select statement.

```
*
  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
```

Controlled Processing

All valid CA-IDMS functions can be performed using the controlled processing commands of the IDMS statement. Basically, each of these commands generates a call to CA-IDMS in much the same manner as a COBOL program. Refer to the appropriate *CA-IDMS DML Reference Guide* for details on the use of these commands. The first parameter of the IDMS statement identifies the command to be issued. These commands are:

```
ACCEPT      BIND
COMMIT      CONNECT
DISCONNECT  ERASE
FIND or OBTAIN  FINISH
GET IF
KEEP MODIFY
READYRETURN
ROLLBACK   STORE
```

You should test the IDMSSTATUS field in the CA-IDMS Communications Block to determine whether the execution of each controlled processing statement is successful.

IDMS Statement

The IDMS statement provides controlled input/output of an CA-IDMS database. You can use the commands of the CA-IDMS statement either with or without the automatic input associated with RETRIEVE or SELECT. Exercise caution when combining automatic input and controlled processing to ensure that currency is maintained for automatic input. You can code these statements at any place in a JOB where an I/O statement for any other file can be coded. See the *CA-Easytrieve Language Reference Guide* for complete syntax for all IDMS statements.

Controlled Processing Examples

The examples that follow use the CA-IDMS test database supplied with your CA-IDMS system.

Area Sweep for Record Type

```

%IDMSCUST. * INVOKE FIELD DEFINITIONS
*
JOB INPUT(NULL), START(SIGN-ON), NAME(AREA-SWEEP)
*
* RETRIEVE FIRST RECORD
IDMS OBTAIN FIRST, RECORD 'CUSTOMER', AREA 'CUSTOMER-REGION'
LOOP
* IF STATUS IS OK, PRINT CUSTOMER NAME
* AND RETRIEVE NEXT RECORD
PERFORM STATUS-CHECK
PRINT AREA-SWEEP
IDMS OBTAIN, NEXT, RECORD 'CUSTOMER', AREA 'CUSTOMER-REGION'
GO TO LOOP
*
SIGN-ON. PROC
*
* BIND THE RUN-UNIT
IDMS BIND 'DEMOSS03'
PERFORM STATUS-CHECK
* ASSIGN RECORD WORK AREA
IDMS BIND, FILE DBASE, RECORD CUSTOMER
PERFORM STATUS-CHECK
* READY THE AREA
IDMS READY, AREA 'CUSTOMER-REGION'
PERFORM STATUS-CHECK
END-PROC
*
STATUS-CHECK. PROC
* IF STATUS NOT OK, TERMINATE PROCESSING
IF IDMSSTATUS NE '0000'
DISPLAY NEWPAGE, 'IDMS STATUS IS ', IDMSSTATUS
IDMS FINISH
STOP
END-IF
END-PROC
*
REPORT AREA-SWEEP LINESIZE(72)
TITLE 'SWEEP OF 'CUSTOMER-REGION' FOR ALL 'CUSTOMERS'''
LINE CUST-NAME

```

Record Retrieval Using a Tickler File

```

%IDMSCUST. * INVOKE FIELD DEFINITIONS
*
FILE KEYFILE
CUSTOMER-KEY 1 10 A
*
JOB INPUT(KEYFILE), START(SIGN-ON), FINISH(SIGN-OFF)
* ESTABLISH KEY AND RETRIEVE RECORD
CUST-NO = CUSTOMER-KEY
IDMS OBTAIN, CALC, RECORD 'CUSTOMER'
* IF "RECORD-NOT-FOUND", INDICATE SO
IF IDMSSTATUS EQ '0326'
CUST-NAME = 'NOT FOUND'
CUST-NO = CUSTOMER-KEY
ELSE
PERFORM STATUS-CHECK
END-IF
* PRODUCE REPORT
PRINT CALC-RPT
*

```

```

SIGN-ON. PROC
*          BIND THE RUN-UNIT
  IDMS BIND 'DEMOSS03'
  PERFORM STATUS-CHECK
*          ASSIGN RECORD WORK AREA
  IDMS BIND, FILE DBASE, RECORD CUSTOMER
  PERFORM STATUS-CHECK
*          READY THE AREA
  IDMS READY, AREA 'CUSTOMER-REGION'
  PERFORM STATUS-CHECK
END-PROC
*
STATUS-CHECK. PROC
*          IF STATUS NOT OK, TERMINATE PROCESSING
  IF IDMSSTATUS NE '0000'
    DISPLAY NEWPAGE, 'IDMS STATUS IS ', IDMSSTATUS
    IDMS FINISH
  END-IF
END-PROC
*
SIGN-OFF. PROC
*          SIGN-OFF THE DATA BASE
  IDMS FINISH
END-PROC
*
REPORT CALC-RPT LINESIZE(72)
  TITLE 'RECORD RETRIEVAL BY CALC KEY'
  LINE CUST-NO CUST-NAME
*
```

Locate all Customer Orders

```

%IDMSCUST. * INVOKE FIELD DEFINITIONS
*
JOB INPUT(NULL), START(SIGN-ON)
*          RETRIEVE FIRST CUSTOMER
  IDMS OBTAIN, FIRST, RECORD 'CUSTOMER', AREA 'CUSTOMER-REGION'
  GO TO CUSTOMER-CHECK
CUSTOMER-NEXT.
  PERFORM STATUS-CHECK
*          RETRIEVE NEXT CUSTOMER
  IDMS OBTAIN, NEXT, RECORD 'CUSTOMER', AREA 'CUSTOMER-REGION'
  IF "END-OF-AREA", SIGN-OFF
  IF IDMSSTATUS EQ '0307'
    IDMS FINISH
    STOP
  END-IF
CUSTOMER-CHECK.
  PERFORM STATUS-CHECK
*          RETRIEVE FIRST ORDER
  IDMS OBTAIN, NEXT, RECORD 'ORDOR', SET 'CUSTOMER-ORDER'
  IF IDMSSTATUS EQ '0307'
    MOVE SPACES TO ORD-NO ORD-CPO#
    PRINT CUST-ORD
    GO TO CUSTOMER-NEXT
  END-IF
  PERFORM STATUS-CHECK
ORDER-NEXT.
  PRINT CUST-ORD
*          RETRIEVE NEXT ORDER AND PRINT
  IDMS OBTAIN, NEXT, RECORD 'ORDOR', SET 'CUSTOMER-ORDER'
  IF IDMSSTATUS EQ '0000'
    GO TO ORDER-NEXT
  END-IF
  IF "END-OF-AREA", GET THE NEXT
  CUSTOMER
  IF IDMSSTATUS EQ '0307'
    GO TO CUSTOMER-NEXT
  END-IF
  PERFORM STATUS-CHECK
*
SIGN-ON. PROC
*          BIND THE RUN-UNIT
```

```
IDMS BIND 'DEMOSS03'
PERFORM STATUS-CHECK
*
IDMS BIND, FILE DBASE, RECORD CUSTOMER
PERFORM STATUS-CHECK
*
IDMS BIND, FILE DBASE, RECORD ORDOR
PERFORM STATUS-CHECK
*
IDMS READY, AREA 'CUSTOMER-REGION'
PERFORM STATUS-CHECK
*
IDMS READY, AREA 'ORDER-REGION'
PERFORM STATUS-CHECK
END-PROC
*
STATUS-CHECK. PROC
*
IF IDMSSTATUS NE '0000'
    DISPLAY NEWPAGE, 'IDMS STATUS IS ', IDMSSTATUS
    IDMS FINISH
    STOP
END-IF
END-PROC
*
REPORT CUST-ORD LINESIZE(72), DTLCTL(FIRST)
SEQUENCE CUST-NO
CONTROL FINAL NOPRINT, CUST-NO NOPRINT, CUST-NAME NOPRINT
TITLE 'CUSTOMER ORDERS'
LINE CUST-NO, CUST-NAME ORD-NO, ORD-CPO#
```

Obtain Logical Record

```

      PARM DEBUG (DMAP)
*
      IDD SUBSCHEMA DEMOSSLR SCHEMA DEMOSCHM +
        SELECT (CUST-SALES-LR)
*
      FILE IDS   FB (80 8000)
      IDENT      1      4      N
*
      JOB INPUT IDS  NAME (LROBTAIN) +
        START SIGN-ON  FINISH SIGN-OFF
*
      IDMS OBTAIN NEXT RECORD CUST-SALES-LR +
        WHERE (IDENT = CUST-NUMBER)
      PERFORM LR-STATUS-CHECK
*
      IF LR-STATUS = 'LR-FOUND'
        DISPLAY CUST-NAME +2 SLS-CUST-NO +2 PROD-DESC
      END-IF
*
      SIGN-ON. PROC
        IDMS BIND 'DEMOSSLR'  DICTNAME 'DICTDB'
        PERFORM STATUS-CHECK
        IDMS READY
        PERFORM STATUS-CHECK
      END-PROC
*
      SIGN-OFF. PROC
        IDMS FINISH
        PERFORM STATUS-CHECK
      END-PROC
*
      STATUS-CHECK. PROC
        IF IDMSSTATUS NE '0000'
          DISPLAY 'IDMS STATUS ' IDMSSTATUS
          DISPLAY 'ERROR DATA: AREA ' IDMSEAREA +
            ' SET ' IDMSESET ' RECORD ' IDMSEREC
          STOP
        END-IF
      END-PROC
*
      LR-STATUS-CHECK. PROC
        IF LR-STATUS EQ 'LR-ERROR'
          DISPLAY 'LR STATUS ' LR-STATUS ' NAME ' LRNAME +
            ' VERB ' LRVERB
          PERFORM STATUS-CHECK
        END-IF
      END-PROC

```

Introduction

The IMS/DL/I interface provides complete facilities for information retrieval and maintenance of IMS/DL/I databases. To use this interface efficiently, you should have a basic knowledge of IMS/DL/I and of the database(s) to be processed. Preparatory work by the database administrator significantly reduces the effort of writing programs that process databases.

The database administrator should place the data definition statements necessary to process each database into the CA-Easytrieve macro library. Control of these segment and field definition statements can greatly reduce the number of simple programming errors associated with database processing.

This chapter discusses CA-Easytrieve database processing requirements in detail. The four CA-Easytrieve statements that define database activities are described in the *CA-Easytrieve Language Reference Guide*.

- FILE statement - identifies the database.
- RECORD statement - identifies the database segments that are available for processing.
- RETRIEVE statement - describes automatic database input.
- DLI statement - provides controlled processing for the creation, retrieval, and maintenance of a database.

CA-Easytrieve cannot access PSBs generated with a language type of PL/I; a language type of ASSEM or COBOL is required.

This chapter also discusses how automatic input uses the RETRIEVE statement in one of three ways:

- Sweep of a database
- Tickler file control
- Input definition (paths).

Test Database

The source statement samples that follow show database definition statements (DBD) and program specification block (PSB) statements that describe the database referenced throughout this chapter. The database is a portion of the PARTS test database provided by IBM with the IMS system. Detailed information about the database can be found in the IBM publication *IMS/VS Installation Guide*. The test database for DLI DOS/VS is described in the IBM publication *Guide for New Users*. This chapter uses only the OS/IMS test database that is shown in the Test Database Structure diagram.

DBD Source Statements

```

DBD      NAME=DI21PART, ACCESS=(HISAM, ISAM)
DATASET DD1=DI21PART, DEVICE=3330, OVFLW=DI21PARO
SEGM     NAME=PARTROOT, PARENT=0, BYTES=50, FREQ=250
FIELD    NAME=(PARTKEY, SEQ), TYPE=C, BYTES=17, START=1
SEGM     NAME=STANINFO, PARENT=PARTROOT, BYTES=85, FREQ=1
FIELD    NAME=(STANKEY, SEQ), TYPE=C, BYTES=2, START=1
SEGM     NAME=STOKSTAT, PARENT=PARTROOT, BYTES=160, FREQ=2
FIELD    NAME=(STOCKEY, SEQ), TYPE=C, BYTES=16, START=1
SEGM     NAME=CYCCOUNT, PARENT=STOKSTAT, BYTES=25, FREQ=1
FIELD    NAME=(CYCLKEY, SEQ), TYPE=C, BYTES=2, START=1
SEGM     NAME=BACKORDR, PARENT=STOKSTAT, BYTES=75, FREQ=0
FIELD    NAME=(BACKKEY, SEQ), TYPE=C, BYTES=10, START=1
DBDGEN

```

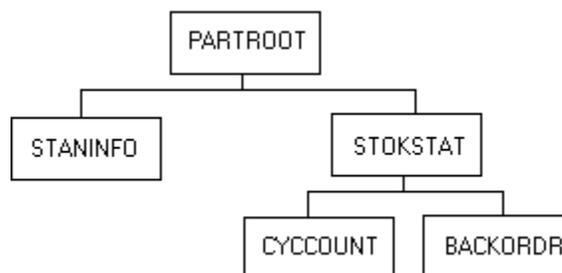
PSB Source Statements

```

PCB      TYPE=DB, DBDNAME=DI21PART, PROCOPT=A, KEYLEN=43
SENSE    PARTROOT
SENSE    STANINFO, PARTROOT
SENSE    STOKSTAT, PARTROOT
SENSE    CYCCOUNT, STOKSTAT
SENSE    BACKORDR, STOKSTAT
PSBGEN   LANG=ASSEM, PSBNAME=EZTPPSBA

```

Test Database Structure



PCB and PSB Processing

CA-Easytrieve uses the Call DLI interface to access an IMS/DLI database. ASMTDLI is statically linked with your CA-Easytrieve program when you run a batch compile. See the *CA-Easytrieve/Online User Guide* for more information about program linkage.

PCB Specification and Access

Regardless of the execution environment, the PCB to be processed is specified on the FILE statement. See the *CA-Easytrieve Language Reference Guide* for a complete description of the FILE statement. Field definitions immediately following the FILE statement map the specified PCB. All PCB fields can be accessed in a CA-Easytrieve program.

PSB Specification

In batch and TSO environments, the PSB name is passed to IMS/DLI as a parameter on the statement that executes the DFSRRC00 program. When your CA-Easytrieve program begins executing, the PCB specified on the FILE statement is found in the PSB by the CA-Easytrieve library routines and made available to your program automatically.

In CICS, a PSB must be explicitly scheduled by your program before any DLI file can be accessed. The DLI PCB statement must be used to schedule a PSB. See the *CA-Easytrieve Language Reference Guide* for a complete description of the DLI statement. Also see the *CICS Application Programmer's Reference Manual* for information about PSB scheduling and terminating. After a DLI PCB statement is executed, all DLI files in the CA-Easytrieve program are accessible. The DLI PCB statement must be executed even when automatic input (the RETRIEVE statement) is being used to read the database. Because the PSB can only be scheduled one time, the DLI PCB statement should be placed in a JOB START proc, or some other one-time only code. When all DLI processing is finished, the DLI TERM statement should be used to terminate the PSB. If this is not done, the PSB remains scheduled until task termination. In non-CICS environments, the DLI PCB and DLI TERM statements have no effect.

Status Information

After each DLI operation, the PCB Status Code field is placed in the CA-Easytrieve system-defined field, FILE-STATUS. See the *IBM DLI Programmer's Reference Manual* for the definition of the status code values.

In CICS, two additional system-defined fields are supplied to give additional status data: UIBFCTR and UIBDLTR. Each is a one-byte binary field. The value of each field is copied directly from the same-named UIB fields after each DLI operation. See the *IBM CICS Application Programmer's Reference Manual* for the definition of the UIB field values.

In non-CICS environments, UIBFCTR and UIBDLTR contain zeros.

Automatic Input

Automatic input is a facility whereby CA-Easytrieve retrieves records from the database and makes them available to the program. See *Controlled vs. Automatic Processing* in the “File Processing” chapter for a description of automatic input as it applies to conventional files. To indicate that automatic input is to occur for the IMS/DLI database, you must do the following:

1. Code the INPUT parameter on the JOB statement for each activity that will require automatic input from the database. The INPUT parameter must specify the filename from the FILE statement that defines the IMS/DLI file. This must be the only filename specified. CA-Easytrieve does not support synchronized file processing for IMS/DLI database files.
2. Code a RETRIEVE statement following the JOB statement. Only one RETRIEVE statement must be coded.

The RETRIEVE statement with SELECT parameters describes the particular portion of the database to be retrieved. CA-Easytrieve performs all the DLI calls needed to retrieve the records described by the automatic input statements. Automatic input is either a sweep of the entire database or a selection of root statements through the use of a tickler file.

Sweep of Database

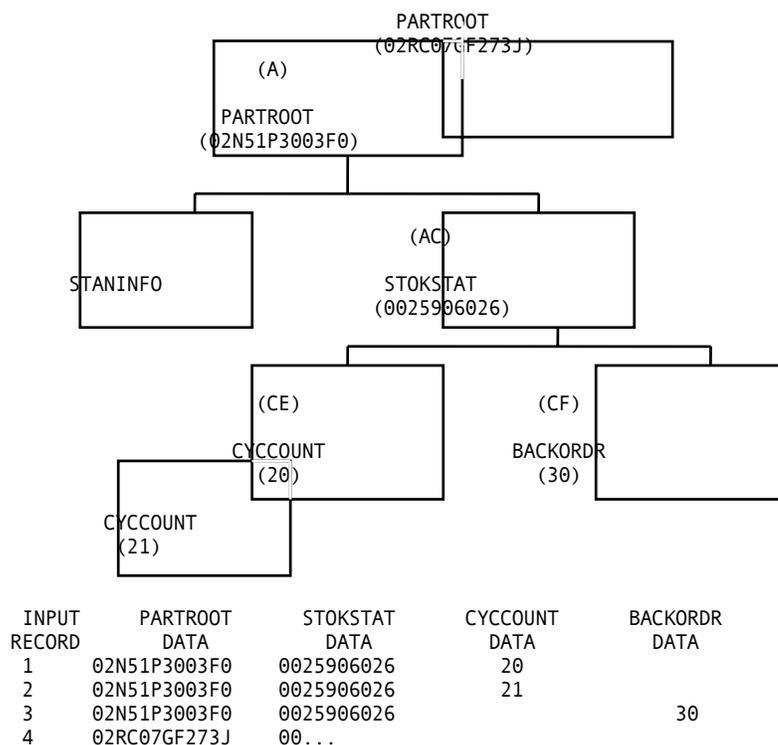
Sweeping the entire database provides the default input. A get next (GN) call is issued at the root level until the database has been exhausted. LIMIT, SSA, or WHILE parameters, if specified, control the sweep.

Tickler File Control

Optionally, a file of root segment keys can control the extent of the database to be processed. Root segment keys are obtained one-at-a-time from the tickler file. Get unique (GU) calls are issued for each key on the tickler file. The KEY parameter must be specified on the RECORD statement for the root segments retrieved by the tickler file option.

Input Definition (Paths)

Automatic input of IMS/DLI databases depends upon the concept of path processing. Each database path identified with the SELECT parameter is processed in a top-to-bottom, front-to-back, and left-to-right order. A root segment is accessed first, with path accessing continuing downward to the left until the end of the path is reached. When the end of each path is reached, that data is made available to the program as an input record. The following exhibit illustrates the paths in a portion of the test database. Path-id is enclosed in parentheses () above the field-name:



CA-Easytrieve exhausts each path before proceeding to the next path. When it exhausts the last path, it retrieves the next root and processing begins anew with the leftmost path.

Typical Path Examples

The following series of path processing examples illustrates the functions of the various statements and parameters associated with automatic database input. The examples are based upon the Test Database Structure depicted earlier in this chapter. Each example also relies upon the data definition indicated below:

```
FILE DLIFILE DLI(DI21PART 1)
DBD-NAME      1  8  A
SEG-LEVEL     9  2  A
```

```

STATUS-CODE      11  2  A
PROC-OPTIONS    13  4  A
RESERVE-DL1     17  4  B
SEG-NAME-FB     21  8  A
LENGTH-FB-KEY   29  4  B
NUMB-SENS-SEGS 33  4  B
KEY-FB-AREA     37 43  A
*
RECORD PARTROOT 50 KEY(PARTKEY 1 17)
PARTKEY         1  17  A
PART-NUMBER     1  17  A
PART-DESC       27 24  A
*
RECORD STANINFO 85 PARTROOT KEY(STANKEY 1 2)
STANKEY         1   2  A
STAN-MAKE-DEPT 48  6  A
STAN-MAKE-TIME 62  3  A
*
RECORD STOKSTAT 160 PARTROOT KEY(STOCKEY 1 16)
STOCKEY         1  16  A
STOK-ON-ORDER  106  8  N
STOK-IN-STOCK  114  8  N
*
RECORD CYCCOUNT 25 STOKSTAT
CYCLKEY         1   2  A
*
RECORD BACKORDR 75 STOKSTAT
BACKKEY         1   2  A
BACK-Q1         11 13  A

```

Tickler File Usage Example

This example illustrates path processing using a tickler file to identify the root segments to be processed.

```

FILE DLIFILE DLI (DI21PART 1)
RECORD PARTROOT 50 KEY(PARTKEY 1 17)
PARTKEY         1  17  A
PART-NUMBER     1  17  A
PART-DESC       27 24  A
FILE KEYS
KEY 1 17 A
JOB INPUT (DLIFILE) NAME MYPROG
RETRIEVE DLIFILE +
  KEYFILE KEYS, +
  KEYVALUE KEY, +
  SELECT PARTROOT
IF PATH-ID EQ 'NF'
  DISPLAY 'ROOT NOT FOUND FOR ' KEY
ELSE
  DISPLAY PART-NUMBER PART-DESC
END-IF

```

Segment Selection Example

Segment selection must include complete paths. Therefore, knowledge of the logical structures of a database is mandatory. The following exhibits illustrate the results of various SELECT parameters.

Root-only Processing

```

...
RETRIEVE DLIFILE ... +
  SELECT (PARTROOT)
...

```

...

Two Path, One Parent Example

```

...
RETRIEVE DLIFILE ... +
  SELECT (PARTROOT +
          STANINFO +
          STOKSTAT)
...
paths are: 1 - PARTROOT AND STANINFO
           2 - PARTROOT AND STOKSTAT

```

Two Path, Two Parent Example

```

...
RETRIEVE DLIFILE ... +
  SELECT (PARTROOT ID 'A' +
          STOKSTAT ID 'AC' +
          CYCCOUNT ID 'CE' +
          BACKORDR ID 'CF')
...
paths are: 1 - PARTROOT AND STOKSTAT AND CYCCOUNT
           2 - PARTROOT AND STOKSTAT AND BACKORDR

```

Path Identification Example

In a typical IMS/DLI database path, each segment can occur multiple times, or it may not occur at all. It is often desirable to be able to determine not only which path is available, but also which segments in the path are available. Based upon the previous Two Path, One Parent Example, the following information is provided:

PATH-ID Value	Segment(s) availability
A	PARTROOT
AC	PARTROOT - STOKSTAT
CE	PARTROOT - STOKSTAT - CYCCOUNT
CF	PARTROOT - STOKSTAT - BACKORDR

Complete Path Processing with Schedule and Terminate

Consider the following example in which we want to process data only from a complete path. That is, when the lowest segment in the path is not present, we want to bypass processing the path altogether.

```

FILE DLIFILE DLI (DI21PART 1)
RECORD PARTROOT 50 KEY(PARTKEY 1 17)
PARTKEY      1 17 A
PART-NUMBER  1 17 A
PART-DESC    27 24 A
RECORD STOKSTAT 160 PARTROOT KEY(STOKKEY 1 16)
STOKKEY      1 17 A
STOK-ON-ORDER 1 17 A
STOK-IN-STOK 27 24 A
RECORD BACKORDR 75 STOKSTAT
BACKKEY      1 2 A
BACK-Q1      11 13 A
JOB INPUT (DLIFILE) NAME MYPROG START INITPSB FINISH TERMP SB
  RETRIEVE DLIFILE +
  SELECT (PARTROOT +
          STOKSTAT +
          BACKORDR ID 'CF')
  IF PATH-ID NE 'CF'
  GO TO JOB
END-IF
DISPLAY PART-NUMBER PART-DESC BACK-Q1
INITPSB. PROC
  DLI PCB 'EZTPPSBA'
END-PROC.
TERMP SB. PROC
  DLI TERM
END-PROC

```

Note: The DLI PCB and DLI TERM statements are required only in CICS environments. They are ignored in non-CICS environments.

Limiting Segment Retrieval

You can use the LIMIT subparameter of SELECT to limit segment occurrence retrieval. The following exhibit describes an area sweep where the number of root segments retrieved is limited to five for program testing purposes. When the limit is reached, input processing is terminated.

```

FILE DLIFILE DLI (DI21PART 1)
RECORD PARTROOT 50 KEY(PARTKEY 1 17)
PARTKEY      1 17 A
PART-NUMBER  1 17 A
PART-DESC    27 24 A
JOB INPUT (DLIFILE) NAME MYPROG
  RETRIEVE DLIFILE +
  SELECT (PARTROOT LIMIT 5)
  DISPLAY PART-NUMBER PART-DESC

```

Another example of limiting segment retrieval is its use to inhibit potential redundant calls to IMS/DLI. For instance, if it is known that a particular segment never occurs more than two times in a path, code LIMIT 2 for that segment. This use of LIMIT improves performance for database activities.

Root Segment Qualification Input Control

You can qualify root segments for retrieval by using the SSA subparameter of SELECT. The value supplied with SSA is enclosed within parentheses and concatenated with the *segment-name* to produce the root segment's SSA. The following example illustrates the control of input through root segment qualification. Processing terminates when IMS/DLI returns a status-code indicating that the qualification cannot be satisfied.

```
FILE DLIFILE DLI (DI21PART 1)
RECORD PARTROOT 50 KEY(PARTKEY 1 17)
PARTKEY      1 17 A
PART-NUMBER  1 17 A
PART-DESC    27 24 A
JOB INPUT (DLIFILE) NAME MYPROG
  RETRIEVE DLIFILE +
    SELECT (PARTROOT SSA 'PARTKEY = 02N51P3003F000  ')
  DISPLAY PARTKEY PART-NUMBER PART-DESC
```

Conditional Segment Retrieval (Segment Pre-screening)

You can pre-screen any segment to establish the acceptability of the segment. CA-Easytrieve bypasses segment occurrences that fail the acceptance test for input consideration. Use the WHILE condition when the Boolean logic of IMS/DLI is inadequate for root SSA qualification or for segment qualification below the root level. The following example demonstrates a segment pre-screen which is unavailable through normal IMS/DLI interfaces.

```
FILE DLIFILE DLI (DI21PART 1)
RECORD PARTROOT 50 KEY(PARTKEY 1 17)
PARTKEY      1 17 A
PART-NUMBER  1 17 A
PART-DESC    27 24 A
JOB INPUT (DLIFILE) NAME MYPROG
  RETRIEVE DLIFILE +
    SELECT (PARTROOT WHILE (PART-DESC ALPHABETIC))
  DISPLAY PART-NUMBER +3 PART-DESC
```

Controlled Processing

You perform IMS/DLI functions using the DLI statement. This statement generates a call to IMS/DLI which is identical to that generated by other programming languages. It is essential that you test the status-code returned in the PCB to determine the success of each DLI statement. These return codes are described in the *IMS/DLI Applications Programming Manual*. The following discussions illustrate typical use of DLI statements.

You must exercise caution when using the DLI statement in conjunction with RETRIEVE. You should save and restore database positioning to ensure the correct continuation of automatic input. Otherwise, input data can be lost or repeated.

When accessing a database in multiple JOBS, it is your responsibility to reposition the database to where processing is to begin in each JOB '2' through JOB 'n'. In the following code for Repositioning Databases with Schedule and Terminate, processing is to begin at the start of the database. By using the START procedure on the JOB statement, DLI statements can be issued to reposition the database at its beginning.

In all the following examples, if the execution environment is CICS, the PSB must be scheduled using the DLI PCB statement before the database is accessed using the DLI or RETRIEVE statement. If the execution environment is not CICS, the DLI PCB and DLI TERM statements have no effect.

```
FILE DLIFILE DLI (DI21PART 1)
RECORD PARTROOT 50 KEY(PARTKEY 1 17)
PARTKEY          1 17 A
PART-NUMBER     1 17 A
PART-DESC       27 24 A
RECORD STOKSTAT 160 PARTROOT KEY(STOCKEY 1 16)
STOCKEY          1 17 A
STOK-ON-ORDER  1 17 A
STOK-IN-STOK    27 24 A
RECORD BACKORDR 75 STOKSTAT
BACKKEY          1 2 A
BACK-Q1          11 13 A
JOB INPUT (DLIFILE) NAME MYPROG START INITPSB
  RETRIEVE DLIFILE +
  SELECT (PARTROOT +
          STOKSTAT +
          BACKORDR ID 'CF')
  IF PATH-ID NE 'CF'
  GO TO JOB
  END-IF
  DISPLAY PART-NUMBER BACK-Q1
  INITPSB. PROC
    DLI PCB 'EZTPPSBA'
  END-PROC
JOB INPUT (DLIFILE) START BEGIN NAME MYPROG2 FINISH TERMPSB
  RETRIEVE DLIFILE +
  SELECT (PARTROOT STOKSTAT ID 'SS')
  IF PATH-ID EQ 'SS'
  DISPLAY STOK-ON-ORDER
  END-IF
*
BEGIN. PROC
  DLI DLIFILE PARTROOT 'GU ' +
  SSA 'PARTROOT(PARTKEY =>999999999999999999)'
  DLI DLIFILE PARTROOT 'GN'
END-PROC
TERMPSB. PROC
  DLI TERM
END-PROC.
```

Complete Path Processing

The following example illustrates the DLI statements necessary to produce the same input data as is produced by the automatic input of the Complete Path Processing discussion earlier in this chapter.

```
SSA-PART      W 37 A VALUE 'PARTROOT(PARTKEY = XXXXXXXXXXXXXXXXXXXX)'
SSA-PART-DATA SSA-PART +19 17 A
SSA-STOK      W 36 A VALUE 'STOKSTAT(STOCKEY = XXXXXXXXXXXXXXXXXXXX)'
SSA-STOK-DATA SSA-STOK +19 16 A
...
JOB INPUT (NULL)
```

```

DLI DLIFILE PARTROOT 'GN'
IF FILE-STATUS EQ 'GE', 'GB'
  STOP
END-IF
IF FILE-STATUS NOT SPACE
  DISPLAY 'JOB TERMINATED, CODE = ' FILE-STATUS
  STOP
END-IF
NEXT-STOK
SSA-PART-DATA = PARTKEY
DLI DLIFILE STOKSTAT 'GNP ' SSA (SSA-PART, +
                                'STOKSTAT ')
IF FILE-STATUS EQ 'GE'
  GO TO JOB
END-IF
IF FILE-STATUS NOT SPACE
  DISPLAY 'JOB ...'
  STOP
END-IF
NEXT-BACK
SSA-STOK-DATA = STOK-KEY
DLI DLIFILE BACKORDR 'GNP ' SSA (SSA-PART, +
                                SSA-STOK, +
                                'BACKORDR ')
IF FILE-STATUS EQ 'GE'
  GO TO NEXT-STOK
END-IF
IF FILE-STATUS NOT SPACE
  DISPLAY 'JOB ... '
  STOP
END-IF
...
process data base path
...
GO TO NEXT-BACK

```

Database Maintenance

You can perform complete database maintenance using the DLI statement. One of the programming techniques used in association with database maintenance is usually the dynamic construction and use of SSAs. The following exhibit (Creation of IMS/DLI Calls) depicts some of the basic programming necessary to dynamically create IMS/DLI calls. The next exhibit (Basic Database Maintenance Activity) shows basic activities that do not require the sophistication of dynamic call generation.

Creation of IMS/DLI Calls

```

...
SSA-COUNT W 4 B
FUNCTION W 4 A
*
SSA-ROOT W 37 A VALUE 'PARTROOT(PARTKEY = XXXXXXXXXXXXXXXXXXXX)'
SSA-ROOT-QUAL SSA-ROOT +8 1 A
SSA-ROOT-DATA SSA-ROOT +19 17 A
*
SSA-LVL2 W 23 A VALUE 'STANINFO(STANKEY = XX)'
SSA-LVL2-SEG SSA-LVL2 8 A
SSA-LVL2-QUAL SSA-LVL2 +8 1 A
SSA-LVL2-KEY SSA-LVL2 +9 8 A
SSA-LVL2-DATA SSA-LVL2 +19 2 A
...
JOB ...
...
SSA-COUNT = 1
SSA-ROOT-QUAL = ' '
FUNCTION = 'GN '
DLI DLIFILE PARTROOT FUNCTION SSANO SSA-COUNT SSA(SSA-ROOT)
PERFORM TEST-STATUS
SSA-ROOT-QUAL = '('
SSA-ROOT-DATA = PARTKEY
FUNCTION = 'GNP '
SSA-LVL2-SEG = 'STANINFO'
SSA-LVL2-QUAL = ' '
SSA-COUNT = 2
PERFORM DLI-CALL
IF FILE-STATUS ...
...
ELSE
PERFORM TEST-STATUS
END-IF
...
DLI-CALL. PROC
DLI DLIFILE STANINFO FUNCTION +
SSANO SSA-COUNT +
SSA (SSA-ROOT, +
SSA-LVL2, +
...)
END-PROC
TEST-STATUS. PROC
IF FILE-STATUS ...
...
END-IF
END-PROC.
...
...

```

Basic Database Maintenance Activity

```
...
...
DLI DLIFILE PARTROOT 'GHN '
IF FILE-STATUS EQ 'GE'
  STOP
ELSE
  PERFORM TEST-STATUS
END-IF
IF PARTKEY ...
  DLI DLIFILE PARTROOT 'DLET'
  PERFORM TEST-STATUS
END-IF
IF FILE-STATUS SPACES
  PRINT
END-IF
...
TEST-STATUS. PROC
  IF FILE-STATUS NOT SPACE
    DISPLAY ...
  ...
  STOP
END-IF
END-PROC
...
...
```


Report Processing

Overview

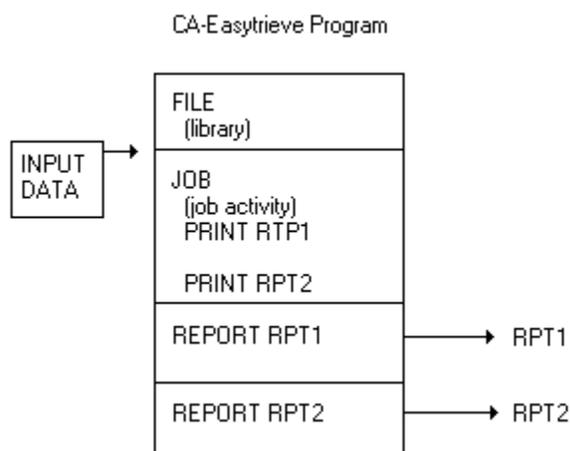
A major function of many CA-Easytrieve programs is to produce printed reports. The non-procedural nature of CA-Easytrieve report syntax is readily adaptable to the production of basic and extremely complex reports, both with minimum programming effort. Two statements generate printed output:

- The PRINT statement initiates the basic declarative report facility.
- The DISPLAY statement produces single print lines on print files.

PRINT is the preferred method because of its many automatic facilities. This chapter primarily discusses report processing using the PRINT statement. Using the DISPLAY statement to mix single print lines within a report is discussed with report procedures.

Basic Report Structure

The CA-Easytrieve report facility is basically declarative; you need only define the format and content of the report and CA-Easytrieve creates the necessary instructions to produce the report. The following exhibit illustrates the basic structure of a CA-Easytrieve job with report processing. You can define one or more reports for each activity.



Note: You can use the CA-Easytrieve/Online Report Painter to automate the coding of a report declaration. See the *CA-Easytrieve/Online User Guide* for more information.

PRINT Statement Processing

The PRINT statement activates the report logic defined by REPORT declarations. CA-Easytrieve extracts the data required for the requested report, formats it in the specified manner, and sends it to the printer. The immediate result of a PRINT statement is either of the following:

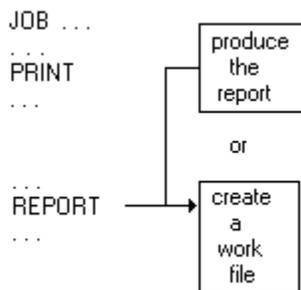
- Data output to a print file
- Data output to a work (or spool) file.

Note: Output to a printer can be sent to a terminal for display (except in UNIX). See Routing Printer Output later in this chapter, for more information.

CA-Easytrieve automatically creates work file records when:

- The report is SEQUENCED
- Another report is already using the associated print file (when you have multiple reports in a single JOB activity).

The results of a PRINT statement are illustrated below:

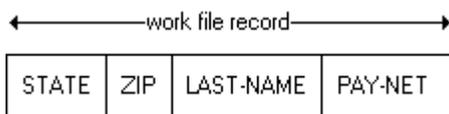


Work File Records

Each work file record contains all of the data required to produce the report. The PRINT statements generate the work file when it is necessary. The order of occurrence of work file fields is the same as the field’s reference occurrence in the REPORT statements. In the next exhibit, the underlined fields determine work file record contents:

```

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
  
```



By default, work files are written to the CA-Easytrieve system work file, *xxxVFM* (where *xxx* is the value of WKDSNPF in the options table). The algorithm for naming work files is *xxxRnnn*, where *xxx* is the value of the WKDSNPF option and *nnn* is the sequential number of the report within the JOB activity. A typical work file name for the first work file in a JOB is EZTR001.

You can save the work file contents for future processing by coding the FILE parameter on the REPORT statement. The corresponding FILE statement, coded in the library, must identify a sequential file. You must also specify that the work file's record length is at least as long as the dynamically created work file record. Records should be blocked to a reasonable value to ensure efficient processing. This technique can also be used to offload the amount of Virtual File Manager (VFM) space required by the program to another file.

Large Reports

If you have multiple large reports in a program, you can code the WORKFILE YES parameter on the PARM statement. This allows you to use a report work file without coding a FILE statement and a FILE parameter on the REPORT statement for every report in your program. You must also add a DD statement for each report to your JCL.

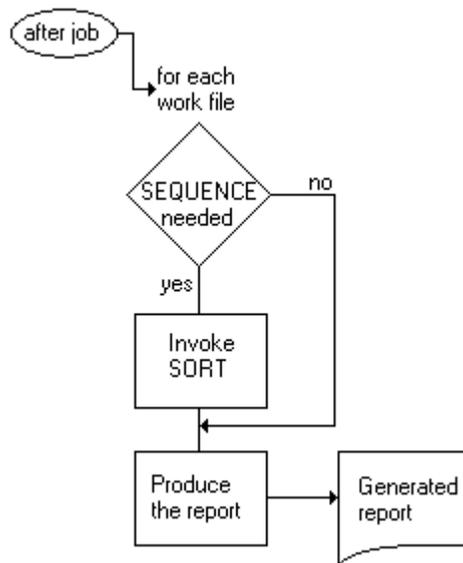
The DDname must conform to the format *xxxRnnn*, where *xxx* is the WKDSNPF option in your site option table (EZT is the default) and *nnn* is the report number. For example, the first report in your program would use the DD EZTR001.

It is also recommended that a large blocksize be coded in the JCL. The default DCB for the work file is FB with record and block lengths set to the minimum needed for the report. You can use the PARM WORKFILE BLOCKMAX parameter to automatically set the blocksize for you. See PARM Statement in the *CA-Easytrieve Language Reference Guide* for more information.

Note: Use VFM for small reports and all reports executing in CICS.

PRINT Workfile Processing

The normal termination process of each JOB activity, illustrated in the following exhibit, includes the processing of any print work files created during the JOB activity. If the JOB activity abends or terminates due to a STOP EXECUTE statement, the print work files are not processed.



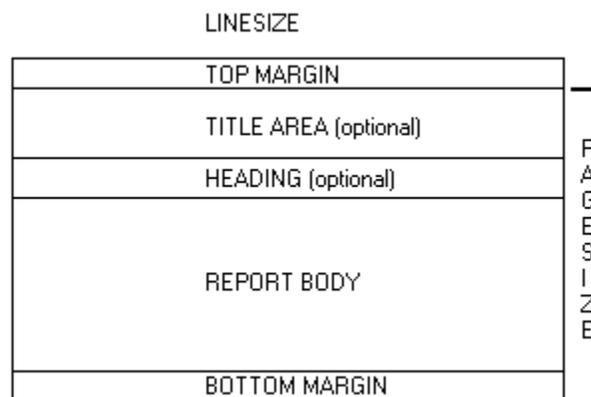
Report Formats

There are two basic report formats:

- Standard format, called a report
- Label format, called a label report.

Standard Format

The CA-Easytrieve default report format is the standard format illustrated below:



Top Margin

The top margin is the space between the physical top of the form and the point to which the printer positions when a top-of-form order is issued to the printer. The size of the top margin is controlled by the printer carriage tape or forms control buffer.

Title Area

The optional title area consists of 1 to 99 title lines plus a title margin between the last title line and the first heading line.

Heading Area

The optional heading area consists of 1 to 99 heading lines plus a heading margin between the last heading line and the report body.

Report Body

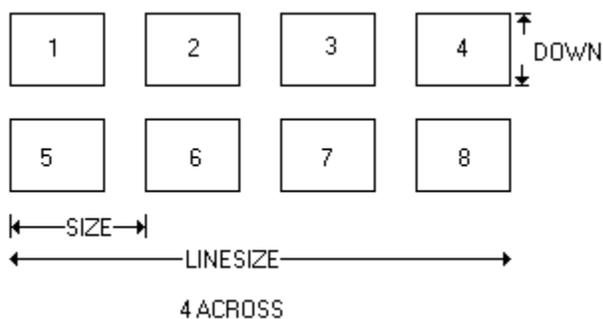
The report body consists of one or more occurrences of a line group. Each occurrence of a line group is 1 to 99 lines plus, optionally, one or more blank lines between occurrences.

Bottom Margin

The bottom margin is the area remaining between the bottom of the report body and the physical bottom of the page.

Label Format

The second report format is used to print labels. The following exhibit illustrates the basic label report page format:



A label line consists of one or more labels positioned across the label page. In the above exhibit, labels one through four compose a label line. A single line group composes each label. CA-Easytrieve produces a label for each PRINT statement execution. CA-Easytrieve formats the labels on the page in the order as numbered above. DOWN and SIZE indicate the dimensions of each label.

REPORT Statement

You define a report in CA-Easytrieve by coding a REPORT statement followed by a series of report definition statements. You must code the REPORT statement first in a report declarative. The REPORT statement establishes the type and characteristics of the report. Although you can specify a large number of REPORT statement parameters, you probably will produce most reports using default parameter values. REPORT statement parameters provide a simple way to define tailored reports. See the *CA-Easytrieve Language Reference Guide* for complete explanations of REPORT statement parameters.

Report Definition Structure

A set of report definition statements defines every CA-Easytrieve report. The statements define the report type, format, sequence, and data content. Report definition statements are the last statements coded in a JOB activity. These statements must be coded in the order illustrated in the following exhibit. You can code report procedures in any order and can define any number of reports for each JOB activity.

	...	
	JOB ...	
	...	
	PRINT ...	
	...	
	REPORT	
	{ SEQUENCE	
	{ CONTROL	
	{ SUM	
Report	{ TITLE	{ REPORT-INPUT
Definition	{ HEADING	{ BEFORE-LINE
Statements	{ LINE	{ AFTER-LINE
	{ special-name procedures	{ BEFORE-BREAK
		{ AFTER-BREAK
		{ ENDPAGE
		{ TERMINATION

Report Definition Statements

The REPORT statement and its parameters define the physical attributes of your report. Code the following statements to define the content of your report:

- SEQUENCE - optionally specifies the order of the report based on the content of one or more fields.
- CONTROL - 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.
- SUM - specifies the quantitative fields which are totaled for a control report.
- TITLE - defines optional report title items and their position on the title line.
- HEADING - optionally defines an alternate heading for a field.

- LINE - defines the content of a report line.

See the *CA-Easytrieve Language Reference Guide* for complete syntax.

System-Defined 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. See Standard Reports for details of line-groups.

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. See Control Reports later in this chapter for more information.

LEVEL

LEVEL indicates the control break level. See Control Reports later in this chapter for more information.

BREAK-LEVEL

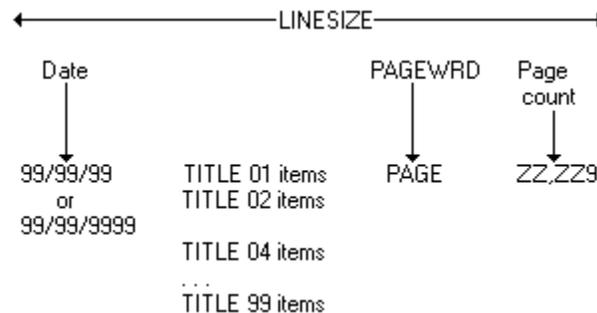
BREAK-LEVEL indicates the highest field in the break. See Control Reports later in this chapter for more information.

Standard Reports

The report facility in CA-Easytrieve includes all of the functions necessary to produce most reports very easily. Using CA-Easytrieve report options, you can produce a report in almost any format. Most reports, however, are variations of a CA-Easytrieve standard report.

Titles

The title is the first item printed on each report page. You can specify the report title with up to 99 TITLE statements. The following exhibit illustrates the title area of a report.



The following are true for standard report titles:

- TITLE 01 items are printed at top-of-form.
- The current date and page count are placed at either end of the TITLE 01 line.
- Title lines are centered within the space indicated by the LINESIZE parameter of the REPORT statement.
- The title line number controls the vertical spacing of titles relative to the first title.
- The SPACE parameter controls the number of blank characters (spaces) between title items. SPACE also controls the separation of the items in the line group in the report body.

Following are title statement examples and their resulting titles:

Statements :

```
FILE PERSNL FB(150 1800)
%PERSNL
JOB INPUT PERSNL NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 LINESIZE 50
TITLE 01 'TEMPORARY EMPLOYEES'
TITLE 03 'IN DEPARTMENT' DEPT
LINE 01 ' '
```

Results:

```

2/09/87      TEMPORARY EMPLOYEES   PAGE      1
              IN DEPARTMENT   903

```

Overriding Defaults

You can override the automatic (default) functions associated with title contents and spacing to produce any desired report title. This may be necessary to produce reports that use pre-printed forms as the output medium. You can use the following parameters to produce non-standard title content and spacing:

- NOADJUST - causes each title line to be left-justified on the page. NOADJUST may cause line items to overlay the tags printed for SUMCTL TAG. COL positioning is necessary for tag to appear.
- NODATE and NOPAGE - inhibit current date and page count information from being placed on the first title line.
- COL - Use the COL positioning parameter to position items in specific print columns. COL can be used with ADJUST or NODADJUST.

Note: The date overlays the left-most positions of TITLE 1 when NOADJUST is specified. Either use NODATE or reserve an area for SYSDATE by specifying COL 10 for SHORTDATE or COL 12 for LONGDATE before the first item on the TITLE statement.

Examples

The following examples of title statements use title content and space adjustment parameters. The report title that results from the statements is also illustrated.

Statements:

```

FILE PERSNL FB(150 1800)
%PERSNL
JOB INPUT PERSNL NAME MYPROG
PRINT REPORT1
REPORT REPORT1 NOADJUST NODATE NOPAGE
TITLE 01 COL 20 SSN
TITLE 02 SYSDATE COL 20 NAME
TITLE 03 COL 20 ADDR-STREET
TITLE 04 COL 20 ADDR-CITY -3 ',' +
-2 ADDR-STATE +5 ADDR-ZIP
LINE 01 ' '

```

Results:

```

column      column
0           2
1           0

11/19/86    025-30-5228
            WIMN   GLORIA
            430 M ST SW 107
            BOSTON   , MA      02005

```

Headings

A report heading is normally printed for line items specified on LINE 01. Each heading is centered over its associated line item. The following rules control the heading; the order in which they are listed indicates the hierarchy of override:

1. The NOHEADING parameter of the REPORT statement inhibits the printing of any headings.
2. The HEADING statement sets the item heading content.
3. The HEADING parameter of the DEFINE statement sets the item heading content.
4. *Field-name* of the DEFINE statement sets the item heading content.
5. Line items which are literals do not have headings.
6. Only LINE 01 items have headings.

The following exhibit illustrates the positioning of headings in a typical report. Line items may not always have the same number of heading entries. In this case, the corresponding heading line area is blank for those items with missing headings.

TITLESKIP space		T I T L E A R E A			
Heading area	HEADING HEADING	HEADING	HEADING	HEADING HEADING HEADING HEADING	
report body	line item	line item	literal line item	line item	line item

The next exhibit illustrates various 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   NAME      8 A
NAME-FIRST  NAME +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 :

	NAME	SOCIAL SECURITY NUMBER		NET PAY
WIMN	GLORIA	025-30-5228	* NO OVERTIME *	251.65
BERG	NANCY	121-16-6413	* NO OVERTIME *	547.88

Line Group

Lines compose the body of a report. The lines of a report are output in groups for each PRINT statement issued. All of the LINE statements of the report make up a line group, which is also called a logical report line.

```
LINE ... }  
LINE 02 ...} line or logical report  
LINE 03 ...} group line  
...
```

Line Item Positioning

Line item positioning follows three rules:

- LINE 01 items and their associated headings are centered in an area whose length is controlled by the longer of the following:
 - The line item
 - The longest heading entry.

The resulting value is called the item length.

- The first line item other than on LINE 01 (that is, LINE 02 through LINE 99) is positioned under the first item of LINE 01. The data is left-justified under the LINE01 data, regardless of the heading size.
- Blank characters (spaces) separate all line items according to the value of the SPACE parameter of the REPORT statement.

Note: When CA-Easytrieve analyzes a LINE statement according to the above rules, the total number of characters on that line must not exceed LINESIZE.

The following exhibit illustrates line item positioning:

```

FILE PERSNL FB(150 1800)
SSN          4 5 P   MASK '999-99-9999' +
                HEADING('SOCIAL' 'SECURITY' 'NUMBER')
EMPNAME      17 20 A   HEADING 'EMPLOYEE NAME'
NAME-LAST   NAME    8 A HEADING('LAST' 'NAME')
NAME-FIRST  NAME +8 12 A HEADING('FIRST' 'NAME')
ADDRESS     37 39 A
ADDR-STREET 37 20 A HEADING 'STREET'
ADDR-CITY   57 12 A HEADING 'CITY'
ADDR-STATE  69 2 A HEADING 'STATE'
ADDR-ZIP    71 5 N HEADING('ZIP' 'CODE')
SEX         127 1 N HEADING('SEX' 'CODE')
JOB INPUT PERSNL NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65
LINE EMPNAME SSN SEX
LINE 02 ADDR-STREET POS 2 ADDR-CITY

      item area      item area      item area
1 5 10 15 1 5 10 1 4
.....

EMPLOYEE NAME      SOCIAL SECURITY NUMBER      SEX CODE      heading
WIMN GLORIA      025-30-5228      1      line
430 M ST SW 107      BOSTON
group

```

Special Positioning

Sometimes the standard positioning of line items on a report is unsuitable for producing the desired result, as in the case of aligning numeric fields on LINE 02 with the decimal point of corresponding fields on LINE 01. The POS line item adjusting parameter left-justifies the corresponding fields, but when the LINE 02 field is not as long as the LINE 01 field, the two fields are unaligned. If that happens, use the *+offset* or *-offset* line item adjustment parameter after the POS parameter to adjust the data's position.

The next exhibit illustrates poor and good decimal alignment. The first occurrence of FLD2 on LINE 02 is not decimal-aligned with FLD1 on LINE 01. To align the second occurrence correctly, an additional *offset* of 3 spaces (+3) is specified.

Statements:

```

DEFINE FLD1 W 4 P 2 VALUE 123.45
DEFINE FLD2 W 3 P 2 VALUE 678.90
JOB INPUT NULL NAME MYPROG
PRINT REPORT1
STOP
*
REPORT REPORT1 LINESIZE 40
LINE 01 FLD1 FLD1
LINE 02 FLD2 POS 2 +3 FLD2
    
```

Results:

	poor		good	
column			column	
1 5 10 15			1 5 10 15	
.....			
FLD1			FLD1	
	123.45		123.45	line 01
	678.90		678.90	line 02

Pre-printed Form Production

Pre-printed form production is another instance when standard line item positioning must be overridden. A very simple example of this override is W-2 form printing in a payroll application. The following code shows the report declarative statements necessary to print a hypothetical W-2 form. This is followed by the result on the pre-printed form.

```

REPORT PAGESIZE 20 NOADJUST NOHEADING SPACE 1
LINE COL 7 'YOUR COMPANY NAME' COL 33 '903' +
COL 39 '12-3456789'
LINE 02 COL 7 'YOUR COMPANY STREET'
LINE 03 COL 7 'YOUR COMPANY CITY STATE ZIP'
LINE 10 COL 7 SSN COL 23 YTD-FEDTAX +
COL 39 YTD-WAGES +
COL 54 YTD-FICA
LINE 12 COL 7 EMP-NAME COL 39 YTD-WAGES
LINE 14 COL 7 EMP-STREET
LINE 15 COL 7 EMP-CITY EMP-STATE EMP-ZIP
    
```

Control NUMBER		State ID NUMBER		FORM W-2	Department		Employer's identification number	
2 Employee's name, address, and ZIP code YOUR COMPANY NAME YOUR COMPANY STREET YOUR COMPANY CITY STATE ZIP				903	12-3456789			
5 Select appropriate boxes		6 Allocated tips		7 Advance EIC payment				
8 Employee's social security number 019-42-0603		9 Federal income tax withheld 3,885.44		10 Wages, tips, other compensation 19,427.20		11 Social security tax withheld 903.88		
12 Employee's name, address, and ZIP code GLORIA WIMN 430 M ST SW 107 BOSTON MA 02005				13 Social security wages 19,427.20		14 Social security tips		
				16		18 Fringe benefits incl. in Box 10		
17 State income tax		18 State wages, tips, etc.		19 Name of State				
20 Local income tax		21 Local wages, tips, etc.		22 Name of locality				

SPREAD Parameter

The SPREAD parameter of the REPORT statement offers a unique way to space line items. When you use reports as work sheets, it is often desirable to space line items as far apart as possible. SPREAD overrides the SPACE parameter of the REPORT statement and creates report lines with the maximum number of spaces between each item, as this exhibit illustrates.

Statements :

```

DEFINE FLD1  W 4 P 2 VALUE 123.45
DEFINE FLD2  W 3 P 2 VALUE 678.90
DEFINE FLD3  W 4 P 2 VALUE 1129.59
JOB INPUT NULL NAME MYPROG
  PRINT REPORT1
  PRINT REPORT2
STOP
*
REPORT REPORT1 SPREAD LINESIZE 65
  TITLE 'S P R E A D  EXAMPLE'
  LINE FLD1 FLD2 FLD3
*
REPORT REPORT2 NOSPREAD LINESIZE 65
  TITLE 'NOSPREAD  EXAMPLE'
  LINE FLD1 FLD2 FLD3

```

Produce :

```

11/19/86          S P R E A D  EXAMPLE          PAGE      1
                  FLD1          FLD2          FLD3
                  123.45        678.90        1,129.59
.....
11/19/86          NOSPREAD  EXAMPLE          PAGE      1
                  FLD1    FLD2    FLD3
                  123.45  678.90  1,129.59

```

Label Reports

You can use the label report capability of CA-Easytrieve to print mailing labels and other applications that require inserting variable data in a repetitious format. A label report is different from a standard report in the following ways:

- Label reports do not have titles and headings.
- Multiple labels can be printed side-by-side.
- Controlled label reports allow for control breaks, but do not automatically total quantitative fields. Totals, however, can be specified on a SUM statement and processed in BEFORE-BREAK and AFTER-BREAK procedures.

You can use the label report function whenever a complete logical print page is to be produced by each PRINT statement. Consider the W-2 form printing example; print time could be substantially reduced by having 2-up forms. You can then modify report declaration statements as shown in the following code. A sample result follows the code.

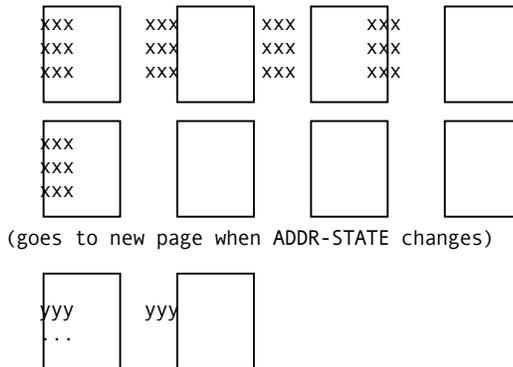
```
REPORT LBL5 LABELS (ACROSS 2 DOWN 15 SIZE 65 NEWPAGE) SPACE 1
LINE 01 COL 7 'YOUR COMPANY NAME' COL 33 '903' +
          COL 39 '12-3456789'
LINE 02 COL 7 'YOUR COMPANY STREET'
LINE 03 COL 7 'YOUR COMPANY CITY STATE ZIP'
LINE 10 COL 7 SSN COL 23 YTD-FEDTAX +
          COL 39 YTD-WAGES +
          COL 54 YTD-FICA
LINE 12 COL 7 EMP-NAME COL 39 YTD-WAGES
LINE 14 COL 7 EMP-STREET
LINE 15 COL 7 EMP-CITY EMP-STATE EMP-ZIP
```

CONTROL Statement

You can use the CONTROL statement with label reports to truncate a group of labels. Truncating makes it easy to separate labels after they are printed. The following exhibit demonstrates how a new label page starts when the control field changes.

Statements :

```
FILE PERSNL FB(150 1800)
%PERSNL
FILE SORTWRK FB(150 1800) VIRTUAL
COPY PERSNL
SORT PERSNL TO SORTWRK +
  USING (ADDR-STATE, ADDR-ZIP) NAME MYSORT
JOB INPUT SORTWRK NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 LABELS
CONTROL ADDR-STATE NEWPAGE
LINE 01 NAME
LINE 02 ADDR-STREET
LINE 03 ADDR-CITY, ADDR-STATE, ADDR-ZIP
```

Results:

Any labels remaining on a line are left unused. The optional NEWPAGE parameter causes a top-of-page for the next print line.

Sequenced Reports

Report sequence is controlled either by the order in which PRINT statements were issued or by the SEQUENCE statement. You can print both standard reports and label reports in any sequence.

Report definitions that contain SEQUENCE statements cause the report data to be spooled to a temporary work file upon execution of a PRINT statement. Work file usage is transparent.

The SEQUENCE function is performed by invoking your installation's sort program or, in CICS, on the workstation, or in UNIX, the sort provided with CA-Easytrieve. The temporary workfile is input to the sort program. When the sort is complete, the work file data is retrieved and the report is produced.

Only those data items used in the report are sorted. The sorted output is directly printed from the sort. These attributes combine to make the SEQUENCE facility of CA-Easytrieve extremely efficient.

CONTROL Reports

The CONTROL statement specifies that a report automatically accumulates and prints totals of quantitative report information. The report accumulates information according to the hierarchy indicated on the CONTROL statement.

Terminology

The following terms are used throughout the discussion on control reports:

- A **control field** is any field named on a CONTROL statement to establish the hierarchy of a control report.
- A **control break** occurs whenever any field of the control hierarchy changes value.
- A **total line** is a logical line group in a report body which contains control totals. These control totals can contain subtotal or final values. Total lines are normally annotated with the value of control fields according to the SUMCTL parameter of the REPORT statement. This is done by listing the control fields first on the LINE statement.
- A **detail line** is the same line data as in a standard report body line (not a total line). Control fields on detail lines are printed according to the DTLCTL parameter of the REPORT statement. The SUMMARY parameter of the REPORT statement inhibits the printing of detail lines on a control report.
- **Accumulators** are system-created fields which contain totals. Accumulators are created for:
 - All fields designated on the SUM statement
 - All active file or W storage quantitative fields designated on the line group (LINE *nn*) statements, if a SUM statement is not specified. (Quantitative fields are numeric fields with a decimal point designation of 0 through 18.)
- **SUMFILE data** are the contents of control fields and accumulators at each minor control break.

Data Reference

In general, report statements and procedures can reference any field of an active file or working storage. (Some report procedures have minor restrictions which are described with the associated procedure.)

Statements and procedures can directly reference data for detail lines in non-control reports. When control or total fields are referenced, CA-Easytrieve automatically adjusts so that SUMFILE data is used. This assures access to the field actually used in the report. SUMFILE data includes all control fields and ten-byte (18 digit) packed fields for all accumulators. (See Summary File later in this chapter.)

TALLY

TALLY is a system-defined field for control reports. TALLY contains the number of detail records that comprise a control break. You can use TALLY on a LINE statement or you can use it in calculations within report procedures. TALLY is commonly used to determine averages for a control break.

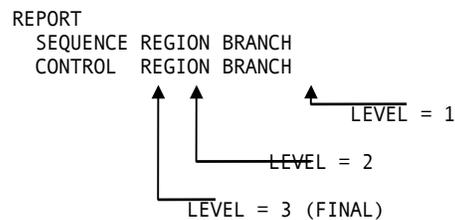
TALLY is a ten-byte packed decimal field with zero decimal places. This definition is used for calculations contained within report procedures. REPORT TALLYSIZE defines the number of digits which are printed for TALLY. A TALLY accumulator is created for each control break level.

LEVEL

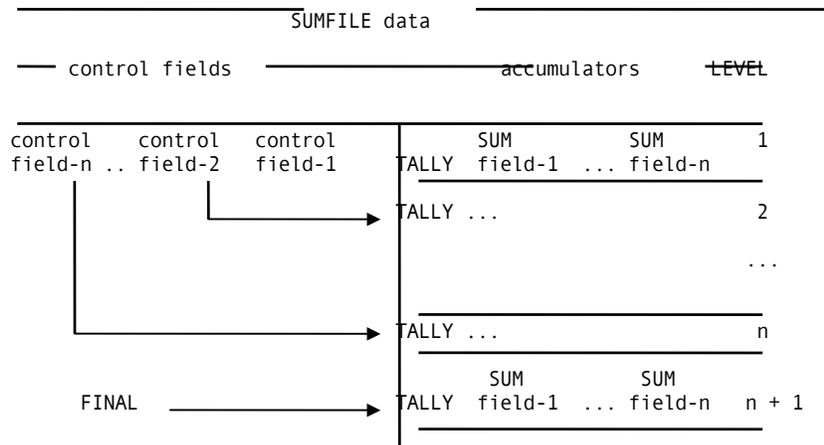
LEVEL is a system-defined field provided for control reports. The field is defined as a two-byte binary field. The value in LEVEL indicates the control break level and varies from 0 to 'n + 1' where:

- LEVEL = 0 when processing detail lines
- LEVEL = n for total line processing at each control level
- LEVEL = n + 1, when processing FINAL totals.

The LEVEL values for fields on the CONTROL statement are illustrated below:



The following exhibit illustrates the relationship between control fields, accumulators, and LEVEL:



See BREAK-LEVEL, which follows, for an example of testing LEVEL and BREAK-LEVEL.

BREAK-LEVEL

BREAK-LEVEL is a system-defined field whose value indicates the highest control break level. The following exhibit illustrates using BREAK-LEVEL to display an appropriate message in a BEFORE-BREAK procedure:

```
REPORT RPT
SEQUENCE REGION BRANCH
CONTROL REGION BRANCH
LINE REGION BRANCH NAME PAY-GROSS
BEFORE-BREAK. PROC
  IF LEVEL = 1
    IF BREAK-LEVEL = 1 . * processing lowest break
      DISPLAY '*** BRANCH TOTALS'
    ELSE-IF BREAK-LEVEL = 2. * region is breaking too
      DISPLAY '*** BRANCH AND REGION TOTALS'
    ELSE-IF BREAK-LEVEL = 3. * final report totals
      DISPLAY '*** BRANCH, REGION, AND FINAL TOTALS'
    END-IF
  END-IF
END-PROC
```

CA-Easytrieve invokes the BEFORE-BREAK procedure before printing summary lines. See Report Procedures later in this chapter for more information.

In the above example, LEVEL and BREAK-LEVEL fields are used to determine the appropriate message to be displayed before the summary lines are printed. Testing for LEVEL 1 tells us that CA-Easytrieve is going to print the first summary line next (BRANCH totals). When BREAK-LEVEL is 1, only the BRANCH field is breaking. Therefore, we want to display a message stating this. When BREAK-LEVEL is 2, the REGION field is breaking. This causes both BRANCH and REGION summary lines to print. When BREAK-LEVEL is 3, CA-Easytrieve prints BRANCH, REGION, and final summary lines.

Note: (Mainframe and UNIX only) An alternative to testing LEVEL and BREAK-LEVEL is to use IF *field-name* BREAK and IF *field-name* HIGHEST-BREAK processing. Using the previous example, you can code the following for the same result:

```
REPORT RPT
SEQUENCE REGION BRANCH
CONTROL REGION BRANCH
LINE REGION BRANCH NAME PAY-GROSS
BEFORE-BREAK. PROC
  IF BRANCH BREAK . * processing lowest break
  IF BRANCH HIGHEST-BREAK . * only branch is breaking
  DISPLAY '*** BRANCH TOTALS ***'
  ELSE-IF REGION HIGHEST-BREAK . * region is breaking also
  DISPLAY '*** BRANCH AND REGION TOTALS ***'
  ELSE-IF REGION HIGHEST-BREAK . * final report totals
  DISPLAY '*** BRANCH, REGION AND FINAL TOTALS ***'
END-IF
END-IF
END-PROC
```

Control Report Contents

The report body contains the only difference between standard and control report contents. Control reports print total lines in addition to detail lines (optional). The following examples use two control fields (STATE and ZIP) which contain data that is two and five bytes long, respectively, and one quantitative field (PAY-NET) which contains numeric data.

The standard control report contains standard report data plus total data. The following example illustrates the report body of such a report. Detail and total lines are shown, with the totals illustrating the hierarchy of the report data.

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
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE ZIP
LINE 01 LAST-NAME STATE ZIP PAY-NET
```

Data:

BROWNIL6007612345
 BROWNIL6007667890
 JONESIL6007709876
 JONESIL6007754321
 SMITHTX7521811111
 SMITHTX7521866666

Results:

<u>Line</u> <u>Description</u>	<u>Control Fields</u>			<u>Accumulator</u>
	LAST-NAME	STATE	ZIP	PAY-NET
detail	BROWN	IL	60076	678.90
detail	BROWN			123.45
ZIP total		IL	60076	802.35
detail	JONES	IL	60077	543.21
detail	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
detail	SMITH	TX	75218	666.66
detail	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

The same report without the detail lines is a SUMMARY report. For example:

Statements:

```
FILE FILE1
LAST-NAME 1 20 A
STATE     21 2 A
ZIP       23 5 N
PAY-NET   28 5 N 2
JOB INPUT FILE1 NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65 SUMMARY +
DTLCTL NONE
SEQUENCE STATE ZIP
CONTROL STATE ZIP
LINE 01 STATE ZIP PAY-NET
```

Data:

```
BROWN IL6007612345
BROWN IL6007667890
JONES IL6007709876
JONES IL6007754321
SMITH TX7521811111
SMITH TX7521866666
```

Results:

<u>Line</u> <u>Description</u>	<u>Control Fields</u>		<u>Accumulator</u>
	STATE	ZIP	PAY-NET
ZIP total	IL	60076	802.35
ZIP total	IL	60077	641.97

STATE total	IL		1444.32
ZIP total	TX	75218	777.77
STATE total	TX		777.77
FINAL total			2222.09

This report contains only control totals because SUMMARY was specified on the REPORT statement.

DTLCTL

The REPORT statement DTLCTL parameter establishes the method for printing control field values on detail lines by using the subparameters EVERY, FIRST, and NONE. The following is an example program using DTLCTL options:

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 option (* with option being EVERY, FIRST, or NONE *)
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE ZIP
LINE 01 LAST-NAME STATE ZIP PAY-NET

```

Data:

```

BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666

```

The next three exhibits show the results of using each of the DTLCTL options:

EVERY -- prints all control fields on every detail line.

<u>Line</u> <u>Description</u>	<u>Control Fields</u>			<u>Accumulator</u>
	LAST-NAME	STATE	ZIP	PAY-NET
detail	BROWN	IL	60076	678.90
detail	BROWN	IL	60076	123.45
ZIP total		IL	60076	802.35
detail	JONES	IL	60077	543.21
detail	JONES	IL	60077	98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
detail	SMITH	TX	75218	666.66
detail	SMITH	TX	75218	111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

FIRST -- prints all control fields on the first detail line at top-of-page and after each break.

<u>Line Description</u>	<u>Control Fields</u>			<u>Accumulator</u>
	LAST-NAME	STATE	ZIP	PAY-NET
detail	BROWN	IL	60076	678.90
detail	BROWN			123.45
ZIP total		IL	60076	802.35
detail	JONES	IL	60077	543.21
detail	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
detail	SMITH	TX	75218	666.66
detail	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

NONE -- prints no control fields on detail lines.

<u>Line Description</u>	<u>Control Fields</u>			<u>Accumulator</u>
	LAST-NAME	STATE	ZIP	PAY-NET
detail	BROWN			678.90
detail	BROWN			123.45
ZIP total		IL	60076	802.35
detail	JONES			543.21
detail	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
detail	SMITH			666.66
detail	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

SUMCTL

The SUMCTL parameter of the REPORT statement establishes the method for printing control field values on total lines of a control report by using the subparameters ALL, HIAR, NONE, and TAG. (The DTLCOPY subparameter controls all non-control non-total values on total lines.) The following shows an example program using these parameters:

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 +
 SUMCTL option
 (* with option being ALL, HIAR, NONE, or TAG *)
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE ZIP
LINE 01 LAST-NAME STATE ZIP PAY-NET

Data:

BROWNIL6007612345
 BROWNIL6007667890
 JONESIL6007709876
 JONESIL6007754321
 SMITHTX7521811111
 SMITHTX7521866666

The next three examples illustrate the results of using All, HIAR, and NONE.

ALL -- prints all control fields on every total line.

Line Description	Control Fields			Accumulator
	LAST-NAME	STATE	ZIP	PAY-NET
	BROWN	IL	60076	678.90
	BROWN			123.45
ZIP total		IL	60076	802.35
	JONES	IL	60077	543.21
	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL	60077	1444.32
	SMITH	TX	75218	666.66
	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX	75218	777.77
FINAL total		TX	75218	2222.09

HIAR -- prints control fields in hierarchical order on total lines.

Line Description	Control Fields			Accumulator
	LAST-NAME	STATE	ZIP	PAY-NET
	BROWN	IL	60076	678.90
	BROWN			123.45
ZIP total		IL	60076	802.35
	JONES	IL	60077	543.21
	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
	SMITH	TX	75218	666.66
	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

NONE -- prints no control fields on total lines.

Line Description	Control Fields			Accumulator
	LAST-NAME	STATE	ZIP	PAY-NET
	BROWN	IL	60076	678.90
ZIP total	BROWN			123.45 802.35
	JONES	IL	60077	543.21
ZIP total	JONES			98.76 641.97
STATE total				1444.32
	SMITH	TX	75218	666.66
ZIP total	SMITH			111.11 777.77
STATE total				777.77
FINAL total				2222.09

TAG

Use the TAG subparameter of SUMCTL to annotate the total line with a description of the total values being printed. The TAG subparameter of SUMCTL creates a line area on the left side of the total line. This LINE 01 item is governed by the following rules:

- The length of the area is the length of the longest control-field-name plus seven.
- TOTAL preceded by the control field name is the annotation for control break totals.
- FINAL TOTAL is the annotation for the final totals line.
- The line item area is positioned at the left margin of the report.

The following example illustrates how tags appear on 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 +
SUMCTL TAG
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE ZIP
LINE 01 LAST-NAME STATE ZIP PAY-NET
```

Data:

```
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
```

JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666

Results:

LAST-NAME	STATE	ZIP	PAY-NET
BROWN	IL	60076	678.90
BROWN			123.45
ZIP TOTAL			802.35
JONES	IL	60077	543.21
JONES			98.76
ZIP TOTAL			641.97
STATE TOTAL			1444.32
SMITH	TX	75218	666.66
SMITH			111.11
ZIP TOTAL			777.77
STATE TOTAL			777.77
FINAL TOTAL			2222.09

DTLCOPY

When printing control reports (particularly a summary report) you may want to include detail information in total lines (normally, CA-Easytrieve prints only control field values and associated totals on total lines). The DTLCOPY subparameter of SUMCTL causes detail field contents (values just prior to the break) to be printed on total lines for LEVEL 1 breaks. The following exhibit illustrates the use of DTLCOPY to print the contents of the LAST-NAME detail field on the lowest level total line (ZIP).

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
```

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
SMITH	TX	75218	777.77
	TX		777.77
			2222.09

DTLCOPYALL

DTLCOPYALL has the same effect as DTLCOPY except that the detail fields are printed for all control break totals. The following exhibit illustrates the use of DTLCOPYALL to print LAST-NAME on all total lines.

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 DTLCOPYALL
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE ZIP
LINE 01 LAST-NAME STATE ZIP PAY-NET

```

Data:

```

BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666

```

Results:

LAST-NAME	STATE	ZIP	PAY-NET
BROWN	IL	60076	802.35
JONES	IL	60077	641.35
JONES	IL		1444.32
SMITH	TX	75218	777.77
SMITH	TX		777.77
SMITH	TX		2222.09

Control Field Values in Titles

Occasionally, you may want to print control field values in report titles. For example, you can use control field annotation within the title of a report to emphasize the structure of an organization, particularly at its higher levels. This technique uses only basic report facilities, and does not require special parameters. The following example shows field annotation within a report title.

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 NEWPAGE ZIP
TITLE 'REPORT FOR THE STATE OF' STATE
LINE 01 LAST-NAME STATE ZIP PAY-NET

```

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   802.35
                JONES      IL     60077   641.97
                JONES      IL     1444.32
11/23/86          REPORT FOR THE STATE OF  TX  PAGE      2
                LAST-NAME  STATE  ZIP    PAY-NET
                SMITH      TX     75218   777.77
                SMITH      TX     777.77
                SMITH      TX     2222.09

```

Overflow of Total Values

In control reports, line items for totaled fields define an area not only for detail lines, but also for corresponding total lines. Since totals are normally larger than the detail, you need a means of expanding the item area. Without this expansion, the item area might be too small to contain the totals. If your report contains this overflow condition, CA-Easytrieve automatically depicts it by setting the right-most character of the item area byte to an * (asterisk), as the following example illustrates:

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 SUMSPACE 0 +
SUMCTL HIAR LINESIZE 65
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE NEWPAGE ZIP
TITLE 'REPORT FOR THE STATE OF' STATE
LINE 01 LAST-NAME STATE ZIP PAY-NET

```

Data:

BROWNIL6007612345
 BROWNIL6007667890
 JONESIL6007709876
 JONESIL6007754321
 SMITHTX7521811111
 SMITHTX7521866666

Results:

```

2/11/87          REPORT FOR THE STATE OF  IL      PAGE 1
                LAST-NAME  STATE   ZIP    PAY-NET
                BROWN      IL      60076  678.90
                BROWN      IL      60076  123.45
                BROWN      IL      60076  802.35
                JONES      IL      60077  543.21
                JONES      IL      60077  98.76
                JONES      IL      60077  641.97
                JONES      IL      60077  444.32*

2/11/87          REPORT FOR THE STATE OF  TX      PAGE 2
                LAST-NAME  STATE   ZIP    PAY-NET
                SMITH      TX      75218  666.66
                SMITH      TX      75218  111.11
                SMITH      TX      75218  777.77
                SMITH      TX      75218  777.77
                SMITH      TX      75218  222.09*
  
```

Controlling Overflow

You can control overflow using two methods, as illustrated by the following examples:

1. *Line Item Expansion* - Ensure that the detail field being totaled is large enough to absorb the totals. The example below illustrates how overflow can be prevented by effectively expanding the line item to six-digit positions.

Statements:

```

FILE FILE1
LAST-NAME 1 5 A
STATE     6 2 A
ZIP       8 5 N
PAY-NET   13 5 N 2
T-PAY-NET W 6 N 2 HEADING ('PAY-NET')
*
JOB INPUT FILE1 NAME MYPROG
T-PAY-NET = PAY-NET
PRINT REPORT1
*
REPORT REPORT1 SUMSPACE 0 +
SUMCTL HIAR LINESIZE 65
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE NEWPAGE ZIP
TITLE 'REPORT FOR THE STATE OF' STATE
LINE 01 LAST-NAME STATE ZIP T-PAY-NET
  
```

Data:

BROWNIL6007612345
 BROWNIL6007667890
 JONESIL6007709876

JONESIL6007754321
 SMITHTX7521811111
 SMITHTX7521866666

Results:

2/11/87 REPORT FOR THE STATE OF IL PAGE 1

LAST-NAME	STATE	ZIP	PAY-NET
BROWN	IL	60076	678.90
BROWN			123.45
	IL	60076	802.35
JONES	IL	60077	543.21
JONES			98.76
	IL	60077	641.97
			1,444.32

2/11/87 REPORT FOR THE STATE OF TX PAGE 2

LAST-NAME	STATE	ZIP	PAY-NET
SMITH	TX	75218	666.66
SMITH			111.11
	TX	75218	777.77
			777.77
			2,222.09

2. *Item Area Expansion* - Expand the item area by using the SUMSPACE parameter of the REPORT statement. The value of SUMSPACE is added to the length of detail fields to determine an adjusted line item length for the total field. The resulting line item expansion is illustrated in the next example as a print edit mask.

Statements:

```
FILE FILE1
LAST-NAME 1 5 A
STATE      6 2 A
ZIP        8 5 N
PAY-NET   13 5 N 2 .* (999.99- mask without SUMSPACE specified)
*
JOB INPUT FILE1 NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 SUMSPACE 1 +
SUMCTL HIAR LINESIZE 65
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE NEWPAGE ZIP
TITLE 'REPORT FOR THE STATE OF' STATE
LINE 01 LAST-NAME STATE ZIP PAY-NET
```

Data:

```
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
```

Results:

2/11/87 REPORT FOR THE STATE OF IL PAGE 1

LAST-NAME	STATE	ZIP	PAY-NET
BROWN	IL	60076	678.90
BROWN			123.45
	IL	60076	802.35

```

                JONES      IL      60077  543.21
                JONES      IL      60077  98.76
                IL          1444.32 (9999.99- mask
                                with SUMSPACE 1)
2/11/87      REPORT FOR THE STATE OF TX      PAGE      2
LAST-NAME    STATE    ZIP      PAY-NET
SMITH        TX      75218   666.66
SMITH        TX      75218   111.11
                TX          777.77
                                2222.09 (9999.99- mask
                                with SUMSPACE 1)

```

Summary File

A summary file, which contains all the control and summed field values at each minor break, can be optionally generated during processing of a control report. JOB activities in your program can subsequently process the summary file to provide reports not otherwise available via the standard report facilities of CA-Easytrieve.

You can produce a summary file by defining the file in the library and then referencing it with the REPORT SUMFILE parameter.

The FILE statement must contain the file name, record format, logical record length, and blocksize. When you just want to retain the file for the duration of the program, you can specify the file as an unblocked VIRTUAL file. The record format can be any standard format. The record length must be large enough to contain the data which is output. Blocksize should be appropriate for the specified format and record length.

The summary file record contains three parts:

1. **Control field area** - a concatenation of the control fields specified on the CONTROL statement. The sum of the lengths of the control fields defines the length of the control field area.
2. **TALLY** - a ten-byte field.
3. **Summed field area** - a concatenation of summed fields to form the remaining segment of the summary file record. Each summed field is a ten-byte packed field with the same decimal specification as the source field.

Therefore, the summary file record length is the sum of the control field area length, plus 10 bytes for TALLY, plus 10 times the number of summed fields.

SUMFILE Example

The following example illustrates the use of SUMFILE data. The values of SFILE are listed in order of ascending magnitude within SFILE-STATE, without reprocessing the original data.

Statements:

```

FILE FILE1
LAST-NAME 1 5 A
STATE     6 2 A
ZIP       8 5 N
PAY-NET   13 5 N 2
FILE SFIL F(30)
SFILE-STATE 1 2 A
SFILE-ZIP   3 5 N
SFILE-TALLY 8 10 P 0
SFILE-PAY-NET 18 10 P 2
JOB INPUT FILE1 NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65 +
SUMMARY SUMFILE SFIL 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
*
JOB INPUT SFIL NAME MYPROG2
PRINT REPORT2
*
REPORT REPORT2 NOADJUST
SEQUENCE SFILE-STATE SFILE-PAY-NET
LINE 01 SFILE-STATE SFILE-ZIP +
SFILE-TALLY SFILE-PAY-NET

```

Data:

```

BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666

```

Results:

SFILE-STATE	SFILE-ZIP	SFILE-TALLY	SFILE-PAY-NET
IL	60077	2	641.97
IL	60076	2	802.35
TX	75218	2	777.77

Report Procedures

Although REPORT statements meet most of all report requirements, some reports depend upon special data manipulation. Report procedures are routines provided by CA-Easytrieve to satisfy this requirement.

Code report procedures at the end of their associated report. The report processor invokes special-name procedures (such as BEFORE-LINE or AFTER-BREAK) as required.

Special-name Report Procedures

Report procedures are invoked at specific points during the report processing activity. You can determine the specific point in time based on the name of the procedure. The exhibit that follows illustrates the use of the procedures listed below:

- REPORT-INPUT - final screening of report input data. Report data can be selected and/or modified. This procedure is invoked after the PRINT statement is executed or as records are read back from the work file (if used).
- BEFORE-LINE - detail line has not been created or printed. It is typically used to modify the contents of fields or annotate the body of the report before line printing.
- AFTER-LINE - detail line has been printed. It is typically used to annotate the body of the report after each line is printed.
- BEFORE-BREAK - modification of totals before total line printing. It is typically used to calculate averages on control reports.
- AFTER-BREAK - total line has been printed. It is typically used to produce special annotation following total lines on control reports.
- ENDPAGE - at end-of-page body based on pagesize. This procedure can be used to produce footers on each page of the report.
- TERMINATION - at end-of-report. This procedure produces end-of-report information such as hash or other control totals.

```

(REPORT-INPUT - caused by the first PRINT statement)
      5/18/84   PROCEDURE  USAGE      PAGE 1
              STATE  ZIP    PAY-NET
detail (BEFORE-LINE)           IL  60076   678.90
      (AFTER-LINE)
      (REPORT-INPUT - caused by the second PRINT statement)
detail (BEFORE-LINE)           IL  60076   123.45
      (AFTER-LINE)
      (REPORT-INPUT - caused by the third PRINT statement)
total  (BEFORE-BREAK)         IL  60076   802.35
      (AFTER-BREAK)
detail (BEFORE-LINE)           IL  60077   543.21
      (AFTER-LINE)
      (REPORT-INPUT - caused by the fourth PRINT statement)
detail (BEFORE-LINE)           IL  60077    98.76
      (AFTER-LINE)
      (REPORT-INPUT - caused by the fifth PRINT statement)
total  (BEFORE-BREAK)         IL  60077   641.97
      (AFTER-BREAK)
total  (BEFORE-BREAK)         IL           1444.32
      (AFTER-BREAK)
      ...
      ...
      (ENDPAGE)

```

Coding Techniques

Coding report procedures is the same as coding procedures within JOB activities, with the following exceptions:

- You cannot use the following input/output generating statements:


```

DELETE   FETCH
GET      INSERT
POINT    PRINT
PUT      READ
SELECT (SQL)  SQL
UPDATE   WRITE

```
- You cannot use the STOP, TRANSFER, or GOTO JOB statements.
- You cannot PERFORM other procedures from within report procedures.
- Use the DISPLAY statement to perform special report annotations. Use of DISPLAY requires the following extra considerations:
 - You cannot code the DISPLAY statement's *display-file-name* parameter. DISPLAY is only to the associated report.

- You cannot code the HEX parameter of DISPLAY.
- DISPLAY lines are counted and included in the end-of-page determination. DISPLAY statements only cause page breaks when the line count exceeds the *display-page-size* REPORT parameter or DISPLAY TITLE or NOTITLE is used.
- With *display-page-size* specified, using DISPLAY in a BEFORE-LINE procedure could result in a page overflow by one line.

Field Reference

In report procedures, you can reference any field contained in an active file or in working storage. When control or total fields are referenced, CA-Easytrieve automatically adjusts so that SUMFILE data is used. This assures access to the field actually used in the report.

Static Working Storage

Fields contained in S storage exhibit unique properties during report processing. S fields are stored in a static working storage area and are not copied onto report work files. All references to S fields occur at the time the report is actually formatted and printed. Remember, the format and print operation can occur at one of two different times; either immediately upon execution of the PRINT statement or after the processing of work files. With this in mind, you should use S storage fields for:

- Temporary work fields for report procedures
- Line annotations controlled from report procedures
- Grand total values from which you can calculate percentages.

The following example illustrates the use of S fields versus W fields in a report (spooled report):

Statements:

```
FILE FILEA
  INPUT-FIELD      1  5  A
  SFLD-RECORD-COUNT S  2  N
  WFLD-RECORD-COUNT W  2  N
JOB INPUT FILEA
  SFLD-RECORD-COUNT = RECORD-COUNT
  WFLD-RECORD-COUNT = RECORD-COUNT
PRINT RPT
REPORT RPT NOADJUST
  SEQUENCE INPUT-FIELD
  LINE INPUT-FIELD SFLD-RECORD-COUNT WFLD-RECORD-COUNT
```

Data:

```
TESTA
TESTB
TESTC
```

Results:

INPUT-FIELD	SFLD-RECORD-COUNT	WFLD-RECORD-COUNT
TESTA	03	01
TESTB	03	02
TESTC	03	03

The SEQUENCE statement causes the report information to be copied to the report work file when the PRINT statement is executed. The format and print operation is performed when the JOB activity completes. Remember, all references to S fields occur when the actual print and format takes place. Thus, the value of SFLD-RECORD-COUNT on the report is always 3. WFLD-RECORD-COUNT is taken from the report work record and so maintains the value it had at the execution of the PRINT statement.

The same program with the SEQUENCE statement removed produces quite different results as illustrated below in the non-spoiled report:

INPUT-FIELD	SFLD-RECORD-COUNT	WFLD-RECORD-COUNT
TESTA	01	01
TESTB	02	02
TESTC	03	03

In this example, a report work file is not needed. The format and print operation occur upon execution of the PRINT statement and the value of SFLD-RECORD-COUNT is captured with each execution of the PRINT statement.

As seen, in non-spoiled reports, there is little difference between S fields and W fields. The difference lies in how and when spooled reports reference the fields. It is important to understand when to use S fields and when to use W fields. Though the report you may be writing is a non-spoiled report today, a future change to your program may cause the report to become spooled. For example, you might add another report before this one or add a SEQUENCE statement to this report. Both these conditions cause your non-spoiled report to become a spooled report. If you do not define your working storage fields properly, your program may produce incorrect output in the future.

REPORT-INPUT

A REPORT-INPUT procedure selects and/or modifies report input data. This procedure is performed for each PRINT statement (report input). In order to cause the data to continue into report processing, you must execute a SELECT statement for the associated input data. In other words, input which is not SELECTed is bypassed for continued processing.

When the report data has been spooled (because the report was 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 implies that if the report is sequenced, the REPORT-INPUT procedure is invoked after the sort program returns the sorted spool record.

Although you can code the logic within the JOB activity itself, it is occasionally desirable to place the logic in a REPORT-INPUT procedure. The next example illustrates use of the REPORT-INPUT procedure in final report input selection. Only the first record within 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  5 5 N VALUE 00000
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
*
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   678.90
                JONES   IL     60077   543.21
                IL                      1222.11

11/23/86          REPORT FOR THE STATE OF  TX          PAGE 2
                LAST-NAME STATE  ZIP    PAY-NET
                SMITH   TX     75218   666.66
                TX                      666.66
                                1888.77

```

BEFORE-LINE and AFTER-LINE

A BEFORE-LINE procedure is invoked immediately before, and an AFTER-LINE procedure immediately following, the printing of each detail line. BEFORE-LINE procedure is the final chance to modify the data in the detail line detail line before it is printed.

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

BEFORE-BREAK

A BEFORE-BREAK procedure can be used to calculate percentages and average totals. These values must be calculated immediately before printing.

The grand-total for percentage and average calculations is often maintained in S storage. TALLY is typically used as the number of items when calculating averages. The value of LEVEL (a system-defined field) can be used to determine which control break is being processed. The value of BREAK-LEVEL can be used to determine the highest level to break.

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

Statements:

```
FILE FILE1
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  5 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 + .005
END-PROC
```

Data:

```
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
```

Results:

LAST-NAME	STATE	ZIP	PAY-NET	PERCENT
BROWN	IL	60076	802.35	36.11
JONES	IL	60077	641.97	28.89
			1444.32	65.00
SMITH	TX	75218	777.77	35.00
			777.77	35.00
			2222.09	100.00

AFTER-BREAK

An AFTER-BREAK procedure can be used to produce special annotation on control reports. The value of LEVEL (a system-defined field) can be used to determine which control break is being processed. The value of BREAK-LEVEL can be used to determine the highest level to break.

In the next 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
```

Produce:

	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

ENDPAGE

An ENDPAGE procedure can be used to produce page footing information. It is invoked whenever end-of-page is detected. It is typically used to produce page totals or other annotations, as in the following example of page footer 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 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

```

Data:

```

BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666

```

Produce:

```

...
* CONFIDENTIAL - FOR INTERNAL USE ONLY *
.....
...

```

TERMINATION

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. The following is an example of report footing:

Statements:

```

FILE FILE1
LAST-NAME 1 5 A
STATE 6 2 A
ZIP 8 5 N
PAY-NET 13 5 N 2
TOTAL-NET 5 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 NOTITLE
  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

```

Data:

```

BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666

```


The following example shows a program that takes advantage of print routing to multiple logical files:

```

FILE PRINTR1 PRINTER F(121)
FILE PRINTR2 PRINTER F(121)
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
        IF ZIP EQ 60076, 60077
          PRINT REPORT1
        ELSE
          PRINT REPORT2
        END-IF
*
REPORT REPORT1 PRINTER PRINTR1 LINESIZE 120 +
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
*
REPORT REPORT2 PRINTER PRINTR2 LINESIZE 120 +
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

```

Each report in the example is routed to a different logical file.

This CA-Easytrieve facility is an efficient way to separate output to different printer form types such as standard paper, labels, or pre-printed forms.

Reporting to the Terminal

Whenever CA-Easytrieve routes output to the originating terminal on the mainframe or on the workstation, the following events take place:

1. When CA-Easytrieve opens a PRINTER file, a Virtual File Manager (VFM) file is substituted for the terminal. Remember that SYSPRINT is the system-defined PRINTER file.
2. When the file is closed, CA-Easytrieve invokes the Report Display Facility to enable you to browse the printed output.
3. You can print the report being displayed from the Report Display Facility. The output is printed to the device specified by your system administrator as the standard print device.
4. In CA-Easytrieve/Online, runtime errors and any snap dumps produced are also written to a VFM file for display with the Report Display Facility. When you request these to be printed, they are printed to the device specified by your system administrator as the standard error print device. When there is no terminal associated with the task, errors are also written to the same device.

5. The order in which printed output is displayed within the Report Display Facility is determined by the following rules:
 - Runtime errors are displayed at the point the error occurs, but after any pending printer output.
 - The contents of PRINTER files are displayed when the PRINTER file is closed, except for REPORT output and DISPLAY statements in JOB activities with non-spooled reports.
 - Each PRINTER file is closed during the termination of the activity that opened it.
 - The system-defined PRINTER file, SYSPRINT, is opened during the initiation of the program. Therefore, it is closed during termination of the program. All DISPLAYs to SYSPRINT are displayed last. This does not include DISPLAY statements within report procedures or within JOB activities with non-spooled reports.
 - Spooled report output, including any DISPLAYs within report procedures are always displayed at the termination of the JOB activity.
 - Non-spooled report output, including any DISPLAYs within the report procedures or the JOB activity, are displayed at the termination of the JOB activity.
 - If an activity does not produce non-spooled reports, output from DISPLAYs within the activity are displayed at the end of the activity that opened the PRINTER file. This is at the end of the program for displays to SYSPRINT.
 - When a JOB activity terminates, the file's contents are displayed in the following order:
 - Non-spooled reports are displayed in the order in which they were defined. These reports are displayed at the time the activity actually terminates. DISPLAY statement output not associated with a REPORT is displayed, along with non-spooled report output to the same printer file.
 - Spooled reports are displayed in the order in which they were defined. These reports are displayed at the time they are despooled.

See the *CA-Easytrieve/Online User Guide* or the *CA-Easytrieve/Workstation User Guide* for more information on the Report Display Facility.

If you instruct CA-Easytrieve to write output to the terminal and there is no terminal associated with the task, the output is routed directly to the device specified by your system administrator as the standard print device.

Extended Reporting

On the mainframe, CA-Easytrieve uses the CA-PSI Subsystems Reporting Environment to produce reports on print devices so that CA-Easytrieve users do not have to be concerned with environment and device-specific characteristics in printing operations. The use of CA-PSI Subsystems Reporting Environment enables printer set definitions to be shared between other Computer Associates products (such as CA-TELON and CA-Easytrieve/IQ) that use the CA-PSI Subsystems Reporting Environment.

The Reporting Environment provides support for Impact Dot, Ink-Jet, and Electro-Photographic printers. This facility interacts with CA-Easytrieve reporting processing to provide support for additional features allowing CA-Easytrieve to:

- Mix multiple different character sets on the same logical print line. For example, in the same report the extended reporting option can process EBCDIC fields and literals, and data containing DBCS (Double Byte Character Set) format codes. Double Byte Character sets represent writing systems that use more than 256 characters, such as Kanji (Japanese characters).
- Process fields and literals belonging to different fonts. For example, in the same report you can use multiple fonts.

A font is a complete set of images of characters and symbols having common characteristics (for example, style, height, width, weight). In a CA-Easytrieve report, each character within a font must have the same amount of lateral space. That is, CA-Easytrieve supports only fixed-pitch fonts.

CA-Easytrieve automatically formats a report compensating for fields and literals that produce characters of different height and width. CA-Easytrieve automatically calculates the size of elements on title, heading, detail, and summary lines.

The CA-PSI Subsystems printer set definition module (PSIXRPSD) defines the character widths that CA-Easytrieve uses. The module can define the widths as either characters per inch (pitch) or a point size where the term point defines the size of a character as a multiple of 1/72nd of an inch.

- Support control codes in addition to the ANSI paper control codes.
- Support printer files that use non-standard record formats, block sizes, and record lengths.

For detailed information about the CA-PSI Subsystems Reporting Environment, see the *CA-PSI Subsystems Reporting Environment Guide*.

In UNIX, CA-Easytrieve provides a subset of the Extended Reporting environment. See the *CA-Easytrieve for UNIX User Guide* for details.

Reporting Environment Example

Using the Reporting Environment, you can use multiple print fonts in a report. This enables you to highlight fields of special significance. The following output shows you an example of what the Reporting Environment can do. The CA-Easytrieve program that produced the report is shown immediately following the output.

REGION PAY SCALE SUMMARY					
REGION: 1					
BRANCH	DEPT	EMPLOYEE NUMBER	GROSS PAY	NET PAY	DEDUCTIONS
01	903	12267	373.60	251.65	121.95
	918	02200	804.64	554.31	250.33
			1,178.24	805.96	372.28
02	943	11473	759.20	547.88	211.32
	935	00370	554.40	340.59	213.81
			1,313.60	888.47	425.13
03	915	02688	146.16	103.43	42.73
	914	11602	344.80	250.89	93.91
			490.96	354.32	136.64
04	917	11931	492.26	355.19	137.07
	911	11357	283.92	215.47	68.45
	932	11467	396.68	259.80	136.63
	911	01963	445.50	356.87	88.63
			1,618.36	1,187.33	431.03
			4,601.16	3,236.08	1,365.08

END OF REGION 1

Program Example

The following CA-Easytrieve program created the previous report. Note the fields and literals preceded by a pound sign (#) and integer.

```

FILE FILEA
REGION 1 1 N
BRANCH 2 2 N          HEADING ( #5 'BRANCH')          <
EMP# 9 5 N           HEADING ('EMPLOYEE' 'NUMBER')
NAME 17 20 A         HEADING ('EMPLOYEE' 'NAME')
STREET 37 20 A
CITY 57 12 A
STATE 69 2 A
ZIP 71 5 N
NET 90 4 P 2         HEADING ('NET' 'PAY')
GROSS 94 4 P 2       HEADING ('GROSS' 'PAY')
DEPT 98 3 N
DEDUCT W 4 P 2       HEADING ('DEDUCTIONS')
JOB INPUT FILEA NAME BASIC
DEDUCT = GROSS - NET
PRINT REPORT1
REPORT REPORT1 LINESIZE 130 PAGESIZE 45 SUMCTL NONE
SEQUENCE REGION BRANCH
CONTROL REGION NEWPAGE BRANCH
TITLE 01 #3 'REGION PAY SCALE SUMMARY'
TITLE 02 #5 'REGION :' -2 #2 REGION
LINE 01 #5 BRANCH DEPT EMP# GROSS NET DEDUCT
AFTER-BREAK. PROC
IF LEVEL = 2

```

```
      DISPLAY SKIP 3 COL 20 #5 'END OF REGION ' #5 REGION      <  
END-IF  
END-PROC
```

See REPORT Statement in the *CA-Easytrieve Language Reference Guide* for complete syntax.

Printer Support

The CA-PSI Subsystems Reporting Environment supports a variety of printers.

Each printer has its own characteristics, especially with respect to the identification of the font, the presentation of print records, and the distinction between character sets. To support each printer's characteristics, CA-PSI Subsystems uses a printer set definition module. This module defines the type of printer that CA-PSI Subsystems supports, and the font codes that the CA-Easytrieve program supports.

An empty CA-PSI Subsystems printer set definition module (PSIXRPSD) is provided as part of the normal installation of CA-Easytrieve/ESP. Therefore, the CA-Easytrieve reporting mechanism uses the default mode of operation. CA-Easytrieve programs can use the Reporting Environment only after your systems programmer generates the printer set definition module. For information on how to generate the module, see the *CA-PSI Subsystems Reporting Environment Guide*.

The printer set definition module provides support for special formatted files (HTML and RTF) and for the following printers:

- IBM 3800-I, IBM 3800-II
- IBM 3800-III, IBM 3800-VIII
- IBM 3820
- IBM 3200
- XEROX 2700, XEROX 8700, XEROX 9700
- FACOM 6715D, FACOM 6716D
- MELCOM 8250, MELCOM 8270, MELCOM 8290
- HITACHI 8196
- TORAY 8500
- SHOWA INFORMATION SYSTEM SP7, SP8
- MEMOREX 1500/1520

Sample printer definitions are provided in the EZTXMP PIELIB.

Printer Identification

The PRINTER command in the printer set definition module identifies the type of printer(s) that the CA-Easytrieve system uses. A unique extended printer name (*ebstring-1*) identifies each printer in the module. This name not only identifies the characteristics of the printer but also enables you to define up to 256 different font codes for use in CA-Easytrieve programs. The font codes cause fields and literals to be correctly formatted into output lines so that the printer can print them using the correct font sets.

The extended reporting printer name is a parameter (*EXTENDED xrpt-printer*) on the FILE statement of the CA-Easytrieve program. Once you define an extended reporting printer name on a CA-Easytrieve FILE statement, any output, whether it is from a report or from the DISPLAY command, directed to that file, is formatted based on the characteristics defined for that extended reporting printer.

The following exhibit illustrates the CA-Easytrieve syntax supported by the Reporting Environment. Note the EXTENDED keyword on the CA-Easytrieve FILE statement. This keyword enables you to associate the extended reporting printer to a CA-Easytrieve printer file. The exhibit also illustrates the use of the CA-Easytrieve DISPLAY statement within the processing logic of a program to direct print lines to that printer file, thereby taking advantage of the extended reporting facilities that the printer provides.

CA-Easytrieve Coding

```
FILE NEWPTR EXTENDED IBM38002 <=== 1. Same name as the
                                extended reporting
                                printer name in the
                                PSIXRPSD module.
.....
JOB INPUT .....
.....
DISPLAY NEWPTR FIELD1 #2 FIELD2 <=== 2. Output a print line
                                to a printer file
                                associated with an
                                extended reporting
                                printer. Also
                                special use of font
                                2 requested.
```

Font Identification

The next exhibit illustrates the definition of two extended reporting printer names in the printer set definition module. Both definitions are for an IBM 3800 printer, but a different set of fonts is associated with each printer.

The FILE statement in the previous exhibit illustrated the association of an extended reporting printer named IBM38002 to a file called NEWPTR. The DISPLAY statement in previous exhibit illustrates the use of different fonts on the same print line.

The fonts used in the previous exhibit are defined for the extended reporting printer called IBM38002. This definition is illustrated in the following exhibit. As a result, FIELD1 is output at 10 characters per inch (the default as no override was coded) and FIELD2 is output using a font of 15 characters per inch. If a field or literal is to use a different font, then you must precede the field with the character '#' followed by an integer. This integer defines the number of the font in the font table of the extended reporting printer that you are using. The entry must exist for the data type of the field or literal, that is EBCDIC, DBCS, or MIXED. The DBCS data type defines data associated with a Double Byte Character Set (DBCS). Use this data type to output characters for languages such as Japanese, Chinese, Korean, and so on.

Printer Set Definition Module - PSIXRPSD

Code this name on FILE statement [EXTENDED <i>xrpt-printer</i>]	Relative Position of Font coded as # <i>nnn</i> .	
↓	↓	
<i>ebstring-1</i> -----	FONT TABLE _ EBCDIC DATA -----	
IBM38001	default	15 cpi
	1.	10 cpi
	2.	12 cpi
IBM38002	default	10 cpi
	1.	12 cpi
	2.	15 cpi

CA-Easytrieve Printer Definitions

Page Mode Printers

Page mode printers can be addressed by CA-Easytrieve as either absolute (IBM3800C and IBM3800E) coordinates, or relative (IBM3800D and IBM3800F) coordinates. Either coordinate scheme produces the same output on the printed page. Functionally, the IBM3800C and IBM3800D definitions are interchangeable. IBM3800E and IBM3800F are also interchangeable.

The record and block sizes on the CA-Easytrieve FILE statement should be as large as possible to get the most advantage of a page mode printer (3800-3 or 3820). Coding a value of U(32760) would be a good value.

To make a USER3801 printer definition, using absolute coordinates for a single 10 pitch font with 6 LPI, code the printer definition as:

```
Set;
Printer Name('USER3801') Use(System);
Page Startpos ( 0, 0);
Dataset ASA(No) Record(V , 4096) Concatenate(Yes);
Defaults Size( 1, 1) Form( 3168, 2400) Fonts( 1 );
SetVerPos Value( X'2BD304D20000') AbsPos ( 9, 4,B);
SetHorPos Value( X'2BD304C60000') AbsPos ( 9, 4,B);
StartPage Value( X'F10040');
FormatPage Value( X'5A');
FormatPage Value( X'0000D3EE9B000000') RecLength( 1, 4,B)
```

```

RecNumber( 13, 4,B);
Font Number( 1) Width( 24.00) Height( 40.00) Format(EBCDIC);
FctHeader Value(X'2BD303F000');
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');

```

With the JCL:

```

//filename DD SYSOUT=*,CHARS=GS10
FILE filename EXTENDED USER3801 ASA
JOB INPUT NULL
    DISPLAY filename #1 'GS10'
    STOP

```

To make a USER3802 printer definition, using relative coordinates for a 12 pitch font with 8 LPI, code the printer definition as:

```

Set;
Printer Name('USER3802') Use(System);
Page Startpos ( 0, 0);
Dataset ASA(No) Record(V , 4096)
Concatenate(Yes);
Defaults Size( 1, 1) Form( 3168, 2400) Fonts( 1 );
SetVerPos Value( X'2BD304C600002BD304D40000') RelPos ( 21, 4,B);
SetHorPos Value( X'2BD304C80000') RelPos ( 9, 4,B);
StartPage Value( X'F10040');
FormatPage Value( X'5A');
FormatPage Value( X'0000D3EE9B000000') RecLength( 1, 4,B)
RecNumber( 13, 4,B);
Font Number( 1) Width( 20.00) Height( 30.00) Format(EBCDIC);
FctHeader Value(X'2BD303F000');
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');

```

With the JCL:

```

//filename DD SYSOUT=*,CHARS=GS12
FILE filename EXTENDED USER3802 ASA
JOB INPUT NULL
    DISPLAY filename #1 'GS12'
    STOP

```

To make a USER3804 printer definition using relative coordinates for a 15 pitch font with 8 LPI, code the printer definition as:

```

Set;
Printer Name('USER3804') Use(System);
Page Startpos ( 0, 0);
Dataset ASA(No) Record(V , 4096) Concatenate(Yes);
Defaults Size( 1, 1) Form( 3168, 2400) Fonts( 1 );
SetVerPos Value( X'2BD304C600002BD304D40000') RelPos ( 21, 4,B);
SetHorPos Value( X'2BD304C80000') RelPos ( 9, 4,B);
StartPage Value( X'F10040');
FormatPage Value( X'5A');
FormatPage Value( X'0000D3EE9B000000') RecLength( 1, 4,B)
RecNumber( 13, 4,B);
Font Number( 1) Width( 16.00) Height( 30.00) Format(EBCDIC);
FctHeader Value(X'2BD303F000');
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');

```

With the JCL:

```
//filename DD SYSOUT=*,CHARS=GS15
FILE filename EXTENDED USER3804 ASA
JOB INPUT NULL
    DISPLAY filename #1 'GS15'
STOP
```

To make a USER3805 printer definition, using relative coordinates for a 15 pitch font with 10 LPI, code the printer definition as:

```
Set;
Printer Name('USER3805') Use(System);
Page Startpos ( 0, 0);
Dataset ASA(No) Record(V , 4096) Concatenate(Yes);
Defaults Size( 1, 1) Form( 3168, 2400) Fonts( 1 );
SetVerPos Value( X'2BD304C600002BD304D40000') RelPos ( 21, 4,B);
SetHorPos Value( X'2BD304C80000') RelPos ( 9, 4,B);
StartPage Value( X'F10040');
FormatPage Value( X'5A');
FormatPage Value( X'0000D3EE9B000000') RecLength( 1, 4,B)
RecNumber( 13, 4,B);
Font Number( 1) Width( 16.00) Height( 24.00) Format(EBCDIC);
FctHeader Value(X'2BD303F000');
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');
```

With the JCL:

```
//filename DD SYSOUT=*,CHARS=GS15
FILE filename EXTENDED USER3805 ASA
JOB INPUT NULL
    DISPLAY filename #1 'GS15'
STOP
```

To make a USER3806 printer definition, using relative coordinates for a set of 10, 12, 15 and 10 bold pitch fonts with 8 LPI, code the printer definition as follows. The default font is 10 pitch:

```
Set;
Printer Name('USER3806') Use(System);
Page Startpos ( 0, 0);
Dataset ASA(No) Record(V , 4096) Concatenate(Yes);
Defaults Size( 1, 1) Form( 3168, 2400) Fonts( 1 );
SetVerPos Value( X'2BD304C600002BD304D40000') RelPos ( 21, 4,B);
SetHorPos Value( X'2BD304C80000') RelPos ( 9, 4,B);
StartPage Value( X'F10040');
FormatPage Value( X'5A');
FormatPage Value( X'0000D3EE9B000000') RecLength( 1, 4,B)
RecNumber( 13, 4,B);
Font Number( 1) Width( 24.00) Height( 30.00) Format(EBCDIC);
FctHeader Value(X'2BD303F000');
Font Number( 2) Width( 20.00) Height( 30.00) Format(EBCDIC);
FctHeader Value(X'2BD303F001');
Font Number( 3) Width( 16.00) Height( 30.00) Format(EBCDIC);
FctHeader Value(X'2BD303F002');
Font Number( 4) Width( 24.00) Height( 30.00) Format(EBCDIC);
FctHeader Value(X'2BD303F003');
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');
```

With the JCL:

```
//filename DD SYSOUT=*,CHARS=(GS10,GS12,GS15,GB10)
```

In the CA-Easytrieve program, #1 (the default, if not specified) refers to font 1 (GS10), #2 refers to font 2 (GS12), #3 refers to font 3 (GS15), and #4 refers to font 4 (GB10).

```
FILE filename EXTENDED USER3806 ASA
JOB INPUT NULL
  DISPLAY filename #1 'GS10' #2 'GS12' #3 'GS15' #4 'GB10'
  STOP
```

To make an IBM3820 printer definition, using absolute coordinates for a 15 pitch EBCDIC font and a 7.5 pitch DBCS font with 8 LPI, code the printer definition as follows. The default font is 15 pitch EBCDIC and 7.5 pitch DBCS:

```
Set;
Printer Name('IBM3820 ') Use(System);
Page Startpos ( 0, 0);
Dataset ASA(No) Record(V , 4096) Concatenate(Yes);
Defaults Size( 1, 1) Form( 3168, 2400) DBCS(IBM) Fonts( 3, 103, 203);
SetVerPos Value( X'2BD304D20000') AbsPos ( 9, 4,B);
SetHorPos Value( X'2BD304C60000') AbsPos ( 9, 4,B);
StartPage Value( X'F10040');
FormatPage Value( X'5A');
FormatPage Value( X'0000D3EE9B000000') RecLength( 1, 4,B)
  RecNumber( 13, 4,B);
Font Number( 3) Width( 16.00) Height( 30.00) Format(EBCDIC);
  FctHeader Value(X'2BD303F001');
Font Number( 103) Width( 32.00) Height( 30.00) Format(DBCS);
  FctHeader Value(X'2BD303F002');
Font Number( 203) Format( 3, 103);
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');
```

With the JCL:

```
//filename DD SYSOUT=*,CHARS=(GS15,M32F)
```

In the CA-Easytrieve program, #3 (the default, if not specified) refers to EBCDIC font 3 (GS15), #2 refers to DBCS font 103 (M32F). Font 203 is used for mixed EBCDIC and DBCS characters.

```
FILE filename EXTENDED IBM3820 ASA
JOB INPUT NULL
  DISPLAY filename #3 'GS15' #103 D'42F142F242F3' +
    #203 X'F1F20E42F342F40FF5F6'
  STOP
```

CA-Easytrieve drives printers using standard IBM procedures. MVS printer information can be given using the CHARS parameter on the DD statement as described in the above examples, or the information can be specified by using the OUTPUT JCL statement.

For example, you could code the following definition to CA-Easytrieve:

```
//SYSPRINT DD SYSOUT=A,CHARS=(GS10,GB10)
```

or if you have a FORMDEF that has CHARS (and any other pertinent information) you could code:

```
//OUT1 OUTPUT CLASS=A,FORMDEF=formdef
//SYSPRINT DD OUTPUT=OUT1
```

VSE printer information can be given using the CHARS and TRC parameters on the SETPRT statement, or the information can be specified in a Printer-Parameter Member.

For example, you could code the following definition to CA-Easytrieve:

```
// ASSGN      SYSxxx,cuu
*   cuu is the address of the 3800 printer as defined to VSE
// SETPRT     SYSxxx,CHARS=(GS10,GS12,GS15),TRC=Y
```

or if you have a Printer-Parameter Member that contains a PAGEDEF,CHARS (and any other pertinent information) you could code:

```
$$ LST CLASS=x,FNO=fname,LST=cuu
...
// ASSGN      SYSxxx,cuu
```

Line Compatibility Mode Printers

To make a LINE3801 printer definition, using line compatibility for a set of 10, 12, and 15 pitch fonts, code the printer definition as follows. The default font is 10 pitch:

```
Set;
Printer Name('LINE3801') Use(System);
Line OverPrint(Merge, 4) Control(Ansi);
Dataset ASA(Yes) Record(F, 206) Device(Printer) MaxLrecl( 206)
  MaxData ( 204);
Defaults Size( 1 ) Fonts( 1 );
Font Number( 1) Width( 7.20) Format(EBCDIC);
  OpCode Value(X'F0');
Font Number( 2) Width( 6.00) Format(EBCDIC);
  OpCode Value(X'F0');
Font Number( 3) Width( 4.80) Format(EBCDIC);
  OpCode Value(X'F0');
Font Number( 4) Width( 7.20) Format(EBCDIC);
  OpCode Value(X'F1');
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');
```

With MVS JCL:

```
//filenme DD SYSOUT=*,CHARS=(GS10,GS12,GS15),DCB=OPTCD=J
```

With VSE JCL:

```
// ASSGN      SYSxxx,cuu
*   cuu is the address of the 3800 printer as defined to VSE
// SETPRT     SYSxxx,CHARS=(GS10,GS12,GS15),TRC=Y
```

In CA-Easytrieve, CHAR GS10 would be referenced by font number 1 (#1). This is the default font (if no font command is specified). CHAR GS12 would be referenced by font number 5 (#5). CHAR GS15 would be referenced by font number 9 (#9).

```
MVS FILE statement:   FILE filenme EXTENDED LINE3801 ASA
VSE FILE statement:  FILE filenme EXTENDED LINE3801 SYSxxx
JOB INPUT NULL
  DISPLAY filenme #1   'GS10' #2 'GS12' #3 'GS15' #4 'GB10'
  DISPLAY filenme     'GS10' #2 'GS12' #3 'GS15' #4 'GB10'
  STOP
```

The above code displays the literal in the respective font.

To make a LINE3802 printer definition, using line compatibility for a set of 12, 15, and 10 pitch fonts, code the printer definition as follows. The default font is 15 pitch.

```
Set;
Printer Name('LINE3802') Use(System);
Line OverPrint(Merge, 4) Control(Ansi);
Dataset ASA(Yes) Record(F , 206) Device(Printer) MaxLrecl( 206)
MaxData ( 204);
Defaults Size( 1 ) Fonts( 6 );
Font Number( 2) Width( 6.00) Format(EBCDIC);
OpCode Value(X'F0');
Font Number( 6) Width( 4.80) Format(EBCDIC);
OpCode Value(X'F1');
Font Number( 7) Width( 7.20) Format(EBCDIC);
OpCode Value(X'F2');
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');
```

With MVS JCL:

```
//filenme DD SYSOUT=*,CHARS=(GS12,GS15,GS10),DCB=OPTCD=J
```

With VSE JCL:

```
// ASSGN SYSxxx,uuu
* uuu is the address of the 3800 printer as defined to VSE
// SETPRT SYSxxx,CHARS=(GS12,GS15,GS10),TRC=Y
```

In CA-Easytrieve, CHAR GS12 would be referenced by font number 2 (#2). CHAR GS15 would be referenced by font number 6 (#6). This is the default font (if no font command is specified). CHAR GS10 would be referenced by font number 7 (#7).

```
MVS FILE statement: FILE filenme EXTENDED LINE3802 ASA
VSE FILE statement: FILE filenme EXTENDED LINE3802 SYSxxx
JOB INPUT NULL
  DISPLAY filenme #2 'GS12' #6 'GS15' #7 'GS10'
  DISPLAY filenme #2 'GS12' 'GS15' #7 'GS10'
  STOP
```

The above code displays the literal in the respective font.

XEROX Printers

Assume that you define a PDE, we will call it PDE200, using the following fonts:

font name	width (dots)	approx point size	type face
L0112A	22	9	1200
L02BOA	20	9	BOLD
L03BOA	22	7	BOLD
L0412A	20	7	1200
L0512A	30	12	1200
L05SCA	30	12	SCRIPT

The following CA-Easytrieve printer set definition would be used. As coded with the DJDE in the FILE-HEADER, this report will print duplex (DUP=YES, both sides of the paper), and will shift the output on the page (SHI=YES, miss the holes on the paper):

```
Set;
Printer Name('XEROX01') Use(System);
Line OverPrint(Print, 4) Control(Ansi);
Dataset ASA(Yes) Record(F , 212);
Defaults Size( 1 ) Fonts( 1 );
FileHeader Value(
' DJDE FORMAT=PDE200, FONTINDEX=0, OVERPRINT=PRINT, DATA=(1,211)' +
',END;');
Font Number( 1) Width( 22.00) Format(EBCDIC);
OpCode Value(X'F1');
Font Number( 2) Width( 20.00) Format(EBCDIC);
OpCode Value(X'F2');
Font Number( 3) Width( 22.00) Format(EBCDIC);
OpCode Value(X'F3');
Font Number( 4) Width( 20.00) Format(EBCDIC);
OpCode Value(X'F4');
Font Number( 5) Width( 30.00) Format(EBCDIC);
OpCode Value(X'F5');
Font Number( 6) Width( 30.00) Format(EBCDIC);
OpCode Value(X'F6');
EndPrinter;
EndSet;
SAVE OBJECT('PSIXRPSD');
```

To actually use this definition, you would code the following CA-Easytrieve program:

With MVS JCL:

```
//filenme DD SYSOUT=*
```

With VSE JCL:

```
//      ASSGN      SYSxxx, cuu
*      cuu is the address of the 3800 printer as defined to VSE
MVS FILE statement:      FILE filenme EXTENDED XEROX01 ASA
VSE FILE statement:      FILE filenme EXTENDED XEROX01 SYSxxx
JOB INPUT NULL
      DISPLAY filenme #1 'L0112A'
      DISPLAY filenme #2 'L02B0A'
      DISPLAY filenme #3 'L03B0A'
      DISPLAY filenme #4 'L0412A'
      DISPLAY filenme #5 'L0512A'
      DISPLAY filenme #6 'L05SCA'
      STOP
```

Screen Processing

Overview

CA-Easytrieve provides all the facilities necessary to display and receive information from an online terminal. As with other features, the non-procedural nature of CA-Easytrieve provides relief from having to deal with many of the complexities of transaction programming.

Note: Screen processing activities are available only in CA-Easytrieve/Online and CA-Easytrieve/Workstation.

Basic Structure

You use a SCREEN activity to describe and process a screen display. The CA-Easytrieve screen processing facility is basically declarative; you only need to define the format and content of the screen and CA-Easytrieve creates the necessary instructions to send and receive the screen. There are two sections in a SCREEN activity.

1. The screen declaration statements that define the contents of the screen.
2. The optional screen procedures that permit you to code procedural logic to perform file I/O or complex editing of fields displayed on the screen.

The following exhibit illustrates the basic structure of screen processing in an CA-Easytrieve program. You can define one or more screens for each program.

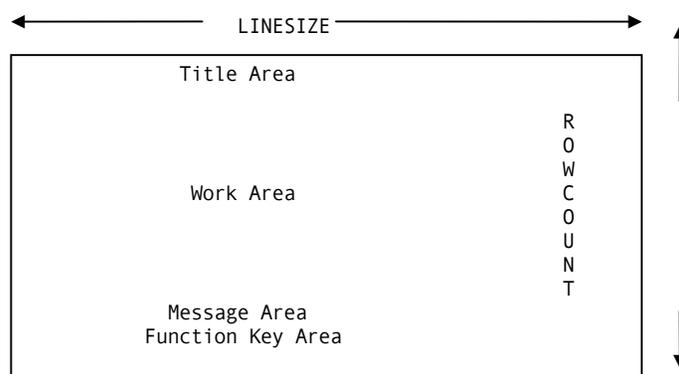
CA-Easytrieve Program

FILE (library section)
SCREEN NAME SCREEN1 Screen Declaration Screen Procedures
SCREEN NAME SCREEN2 Screen Declaration Screen Procedures

Note: The screen declaration process can be automated using the CA-Easytrieve/Online Screen Painter. See the *CA-Easytrieve/Online User Guide* for complete information.

Screen Format

The CA-Easytrieve screen format is illustrated below. The size of the screen defaults to the values specified in your site options. These values can be overridden by the `LINESIZE` and `ROWCOUNT` parameters of the `SCREEN` statement.



Title Area

The title area is an optional area that consists of screen rows designated as titles by `TITLE` statements in the screen declaration. Titles normally identify the screen to the user and are automatically centered at the top of the screen. The title area cannot be updated by the terminal user.

Work Area

The work area contains the items to be displayed to or received from the terminal user. The items are specified by `ROW` statements in the screen declaration. Repeating groups of rows can be specified with the `REPEAT` and `END-REPEAT` statements.

Message Area

The message area is used to display system and programmer-issued messages to the terminal user. The default location of the message area is the line just above the function key display area at the bottom of the screen. You can issue your own messages using the `MESSAGE` or `SET` statement.

Function Key Area

The optional function key area is used to tell the terminal user which function keys are active and the action they perform. This area, if used, is always located on the last line(s) at the bottom of the screen. You use the KEY statement in the screen declaration to define the function key area.

Screen Borders

Specifying the BORDER parameter on a SCREEN statement reduces the size of available screen areas to allow space for the border characters. A border reduces the total screen area by two rows and four columns. For example, if you define a screen with a border and LINESIZE(80) ROWCOUNT(24), available rows are from one to 22, and available columns are from one to 76. ROW 1, COLUMN 1 always refers to the first usable display position on the screen.

Screen Example

The following exhibit illustrates the type of screen that can be created with CA-Easytrieve.

```

Employee File Main Menu

Type an option, then press Enter.
Option ==> W

V View employee
E Edit employee
D Delete employee
X Exit

Please type V, E, D, or X
F1=Help F3=Exit F12=Cancel

```

Following is the screen declaration used to create the example screen above:

```

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
KEY ENTER

```

SCREEN Statement

You define a screen in CA-Easytrieve by coding a SCREEN statement followed by a series of screen definition statements. You must code the SCREEN statement first in a screen declaration. The SCREEN statement defines the characteristics of the screen and activity. See the *CA-Easytrieve Language Reference Guide* for complete explanations of SCREEN statement parameters.

Screen Definition Statements

A set of screen definition statements defines every CA-Easytrieve screen. The statements define the screen format and data content. Screen definition statements must follow the SCREEN statements and precede any procedures in the SCREEN activity. The following exhibit illustrates the order that statements must appear in a SCREEN activity:

```

Screen      SCREEN ...
Definition {  DEFAULT
Statements {  KEY
              {  TITLE
              {  ROW/REPEAT

              special-named procedures {  INITIATION
                                       {  BEFORE-SCREEN
                                       {  AFTER-SCREEN
                                       {  TERMINATION

              programmer-defined procedures

```

- DEFAULT - optionally overrides system-defined screen attributes and message locations.
- KEY - defines valid terminal keys for a screen, specifies descriptive text, and assigns functions to terminal keys.
- TITLE - defines optional screen title items, their attributes, and their position on the title row.
- ROW - defines the contents of a screen row. Item attributes and positioning are optionally specified.
- REPEAT - displays arrays on a screen.

See the *CA-Easytrieve Language Reference Guide* for complete syntax of these statements.

Screen Items

Screen items include the fields and literals that you want to display to or receive from the terminal user. Basic rules regarding items on a screen include the following:

- Unless directed otherwise, CA-Easytrieve automatically places items for the same screen row one space apart. You can optionally add to this space or locate an item at a specific column number.

- The space preceding each item contains system information describing the screen attributes for the item. The attributes contain information that controls the display of screen items such as color and brightness. CA-Easytrieve always uses the space preceding each screen item for attributes.

Note: The space preceding an item located in the first column of any screen row is actually located in the last column of the previous screen row. The space preceding an item located in the first column of the first screen row is located in the last column of the last screen row.

- You can override the automatic placement of items by using the COL parameter. You use COL to specify an explicit screen column number where the item is placed. The COL parameter specifies the column number where the data contained in the item is placed.

Use an offset (+n) to add to the minimum single space used between items (+1 is the default). The offset applies to the data to be displayed. To add additional space, use an offset greater than +1. There must always be at least one space between each item on the screen. Items cannot overlap each other.

- You can specify screen attributes for each item on the screen. If you do not specify attributes for each item, default attributes apply. The following hierarchy is used to determine screen attributes:
 - If attributes are not specified for an item, (ATTR parameter on the ROW or TITLE statement), CA-Easytrieve uses default attributes specified on the DEFAULT statement at the beginning of the SCREEN declaration.
 - If a DEFAULT statement is not coded in the SCREEN activity, CA-Easytrieve uses attributes set in the site options.

Attributes can be specified either as one or more keywords or by using a declared screen attribute. Use of declared attributes provides the ability to define and name a set of attribute keywords. See *Declaring Screen Item Attributes* in “Coding a CA-Easytrieve Program” chapter in this guide, and *DECLARE Statement* in the *CA-Easytrieve Language Reference Guide* for more information. In addition, screen attributes can be changed dynamically during program execution using the SET statement. See *SET Statement* in the *CA-Easytrieve Language Reference Guide*.

- You must ensure that fields used on a screen are in available storage. This requires that you code WORKAREA on FILE statements for fields used on the screen where you do not execute an input statement to fill the fields with data prior to displaying the screen.

Note: You must know the record length to reserve a WORKAREA. CA-Easytrieve does not initialize WORKAREAs. Results are unpredictable if an uninitialized WORKAREA is displayed.

System-Defined 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 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. 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 F2	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
Read Partition		232
No Aid Generated		255

TERM-COLUMNS

TERM-COLUMNS is a two-byte binary field containing 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 two-byte binary 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 16-byte alphanumeric field containing the terminal identification. This field is set only in CICS environments.

SYSUSERID

SYSUSERID is a 16-byte alphanumeric field identifying the terminal user. In CICS, SYSUSERID is copied from the EIB.

Screen Title Area

The title area is the first area on each screen. A screen title is optional but well-designed screens are usually identified with a title. You specify the screen title with TITLE statements coded in the SCREEN declaration.

Title Rules

Following are the rules for specifying screen titles:

- Titles are for display purposes only. You cannot receive data from the terminal user in a TITLE. (To receive data, use a ROW statement in the screen work area.)
- You can specify the screen row number for the title in the TITLE statement. If you do not specify an explicit row number, the next screen row number is assigned. The next row number is one higher than the previous TITLE or ROW statement coded. If no TITLE statement is previously coded, the title is assigned to the top of the screen.
- All title row numbers must precede row numbers associated with the screen work area.
- You need not code TITLE statements for empty rows between titles. For example, if you specify titles for rows 1 and 3 of the screen, row 2 is also considered part of the title area.
- Title items are automatically centered on the screen based on the LINESIZE parameter of the SCREEN statement.

- All title items without explicit column numbers participate in centering.
- Multiple items on a title row are automatically separated from each other by one space. This space contains the screen attributes for the second of the two items. You can optionally add to this space or locate an item at a specific column number.
- You can override the automatic placement of title items by using the COL parameter. You use COL to specify an explicit screen column number where the title item is placed. The COL parameter specifies the column number where the data contained in the item is placed.

Use an offset (+n) to add to the minimum single space used between items (+1 is the default). The offset applies to the title to be displayed. To add additional space, use an offset greater than +1. There must always be the minimum one space between each item on the screen. Title items cannot overlap each other.

- You can specify screen attributes for each title item on the screen. If you do not specify attributes for each item, default attributes apply. The following hierarchy is used to determine screen attributes:
 - If attributes are not specified for a title item, (ATTR parameter on the TITLE statement), CA-Easytrieve uses default attributes specified on the DEFAULT TITLE statement at the beginning of the SCREEN declaration.
 - If a DEFAULT TITLE statement is not coded in the SCREEN activity, CA-Easytrieve uses attributes set in the site options.

Because titles are not updatable, the following attributes are flagged as warnings when compiled, and ignored when used:

```
CURSOR      NUMERIC
INVISIBLE   MUSTFILL
MUSTENTER  TRIGGER
ALARM
```

Title Examples

Following are title statement examples and their resulting screen titles.

Default Centering and Attributes

This example illustrates two title rows that are automatically centered on the screen. The titles are displayed with default screen attributes.

```
SCREEN NAME SCREEN1
  TITLE 1 'Personnel View Utility'
  TITLE 2 'Acme, Inc.'
```

Personnel View Utility Acme, Inc.

Explicit Locations and Attributes

This example shows titles that contain items that are explicitly located on the title row using column specification (COL). The company name in the second title row is displayed bright yellow because the ATTR parameter for the literal is coded to override the default set of attributes for title items.

```
SCREEN NAME SCREEN1
TITLE 1 COL 1 'ViewUtil' 'Personnel View Utility' COL 73 SYSDATE
TITLE 2 'Acme, Inc.' ATTR (INTENSE YELLOW) COL 73 SYSTIME
```

ViewUtil	Personnel View Utility	07/08/90
	Acme, Inc.	12:32:04

Screen Work Area

The screen work area is built by coding ROW statements in a SCREEN declaration. Each ROW statement describes the fields and literals to be located on each row of the screen.

Item Location

The following rules apply to the location of items in the screen work area:

- You can specify the screen row number in the ROW statement. If you do not specify a row number, the next screen row number is assigned. The next row number is one higher than the row number for the previous ROW or TITLE statement. If no TITLE or ROW statement is previously coded, the row is assigned to the first line at the top of the screen.
- All title rows must precede rows associated with the screen work area. This does not mean that you must code all TITLES before ROWs. Instead, a row number explicitly or implicitly defined for a title cannot be greater than any row number defined for a work area row.
- You need not code ROW statements for empty rows between rows with data. You can code ROW statements as placeholders for other ROWs defined without row-numbers. An empty row is coded with a ROW statement without any items.
- Multiple items on a row are automatically separated from each other by one space. This space contains the screen attributes for the second of the two items. You can optionally add to this space or locate an item at a specific column number.
- You can override the automatic placement of items using the COL parameter. You use the COL parameter to specify an explicit screen column number where the item is placed. The COL parameter specifies the column number where the data contained in the item is placed.

Use an offset (+n) to add to the minimum single space used between items (+1 is the default). The offset applies to the data to be displayed. To add additional space, use an offset greater than +1. There must always be at least one space between each item on the screen. Items cannot overlap each other.

- You can repeat a ROW statement or group of ROW statements within the REPEAT and END-REPEAT statements to display an array on a screen.
- You can specify the screen attributes for each item on a work area row. If you do not specify attributes for each item, default attributes apply as follows:
 - If attributes are not specified for a field (ATTR parameter on the ROW statement), CA-Easytrieve uses default attributes specified on the DEFAULT FIELD statement, if coded at the beginning of the SCREEN declaration.
 - If attributes are not specified for a literal (ATTR parameter), CA-Easytrieve uses default attributes specified on the DEFAULT LITERAL statement, if coded.
 - If DEFAULT statements are not coded in the SCREEN activity, CA-Easytrieve uses attributes set in site options.

Because literals and read-only fields are not updatable, the following attributes are flagged as warnings when compiled and ignored when used:

```
CURSOR      NUMERIC
INVISIBLE   MUSTFILL
MUSTENTER   TRIGGER
ALARM
```

- Field attributes can be changed during program execution with the SET statement. See SET Statement in the *CA-Easytrieve Language Reference Guide* for more information.

Location Examples

The following example illustrates various ROW statements and their resulting screen displays.

```
SCREEN NAME SCREEN1
TITLE 1 'Personnel View Utility'
ROW 3 'Type the following information, then press Enter.'
ROW 6 COL 10 'Name . . . .' EMPNAME
ROW 8 COL 10 'Gross Pay . .' GROSS-PAY
ROW 10 COL 10 'Dept . . . .' DEPT-NO
```

```

                                     Personnel View Utility
Type the following information, then press Enter.

Name . . . . BERG
Gross Pay . . 759.20
```

```
Dept . . . . 943
```

Attribute Examples

The following example illustrates various ROW statements coded with specific attributes. The default attribute for fields is changed to protect the data. The screen attribute for GROSS-PAY is then specified to unprotect data entry in the field. CA-Easytrieve automatically places the cursor in the first unprotected field on the screen.

```
SCREEN NAME SCREEN1
DEFAULT FIELD ATTR (PROTECT TURQUOISE)
TITLE 1 'Personnel View Utility'
ROW 3 'Type the new gross pay, then press Enter.' ATTR WHITE
ROW 6 COL 10 'Name . . . . EMPNAME
ROW 8 COL 10 'Gross Pay . . ' GROSS-PAY ATTR (INTENSE TURQUOISE)
ROW 10 COL 10 'Dept . . . . DEPT-NO
```

```

                                Personnel View Utility
Type the following information, then press Enter.

Name . . . . BERG
Gross Pay . . _ 759.20
Dept . . . . 943
```

Note: When you override the default attribute setting, you must supply all of the attributes for the item. Attributes are not merged. For example, if the default attribute is BLUE PROTECT and you specify ATTR TURQ, the field is left unprotected because you did not also specify PROTECT in the override.

Formatting an Item for Display

You can use the following parameters to customize the display of an item in the screen work area:

- FILL
- JUSTIFY
- MASK

Filling an Item for Display

Use the FILL parameter to translate all trailing blanks in a field or literal to a specific character or nulls. If the field is also an input field, CA-Easytrieve automatically translates all fill characters to spaces before placing the data back into the field data area.

Varying length fields with FILL NULL specified do not have trailing nulls translated to spaces. The first trailing null terminates the varying length field and sets its length.

Filling with Underscores

The following example illustrates filling a data entry field with underscores to show the terminal user how much data can be entered. CA-Easytrieve removes any remaining underscores when the screen is received.

```
SCREEN NAME SCREEN1
TITLE 1 'Personnel View Utility'
ROW 3 'Change the employee's name, then press Enter.'
ROW 6 COL 10 'Name . . . .' EMPNAME FILL '_'
```

Personnel View Utility
Change the employee's name, then press Enter.
Name BERG _____

Filling with NULLs

The following example illustrates filling a data entry field with nulls in order to allow the user to insert characters into the field. The 3270 Display Station requires trailing nulls in a field in order for insertion to work.

```
SCREEN NAME SCREEN1
TITLE 1 'Personnel View Utility'
ROW 3 'Change the employee's name, then press Enter.'
ROW 6 COL 10 'Name . . . .' EMPNAME FILL NULL
```

The screen the user receives is:

Personnel View Utility
Change the employee's name, then press Enter.
Name BRG

The user can then insert characters into the field to change it to the correct name, BERG.

Justifying a Field's Contents

CA-Easytrieve normally displays data exactly as it exists in the field as determined by its definition.

The JUSTIFY RIGHT parameter shifts the data in the display field to the right. Trailing spaces and nulls are deleted and leading spaces are inserted. The JUSTIFY LEFT parameter shifts the data in the display field to the left. Leading spaces and nulls are deleted and trailing spaces are inserted. The following example illustrates these two parameters.

```
SCREEN NAME SCREEN1
TITLE 1 'Personnel View Utility'
ROW 3 COL 10 'Name . . . .' EMPNAME
ROW 4 COL 10 'Name . . . .' EMPNAME JUSTIFY RIGHT
ROW 6 COL 10 'Gross Pay . .' GROSS-PAY
ROW 7 COL 10 'Gross Pay . .' GROSS-PAY JUSTIFY LEFT
```

Personnel View Utility	
Name	BERG
Name	BERG
Gross Pay . .	759.20
Gross Pay . .	759.20

Using Edit Masks for Display

CA-Easytrieve automatically applies a mask to numeric fields when displayed. All numeric fields have a default edit mask. You can override this default with a mask you define. See DEFINE Statement and MASK Parameter in the *CA-Easytrieve Language Reference Guide* for an explanation of the default mask and how to code your own mask.

When CA-Easytrieve displays a numeric field, it uses the mask associated with the field (either the default or its override on the DEFINE statement). If you want to use a mask other than the default or override mask for display upon the screen, use the MASK parameter on the ROW statement. The mask on the ROW specifies the mask to be used for this specific occurrence of the field on the screen. If the field is used more than once on the screen, the mask must be specified for each occurrence.

You can use a mask identifier to identify a mask for future use. This shortens coding time when you want to use a particular mask for several fields of the same size on the screen.

If you have defined a mask for a field on a DEFINE statement and you want to use the system-defined default mask, code NOMASK on the ROW statement for the field.

Mask Example

The following exhibit illustrates masks.

```

DEFINE FIELD-WITH-DEFAULT-MASK W 4 P 2 VALUE 1234.56
DEFINE FIELD-WITH-DEFINED-MASK W 4 P 2 VALUE 1234.56 MASK '$$, $$$.99'
DEFINE FIELD-WITH-BWZ-MASK W 4 P 2 VALUE 0 MASK BWZ
SCREEN NAME SCREEN1
  TITLE 1 'Mask Examples'
  ROW 3 'Using Default Mask' FIELD-WITH-DEFAULT-MASK
  ROW 4 'Using Defined Mask' FIELD-WITH-DEFINED-MASK
  ROW 5 'Applying a Mask ' FIELD-WITH-DEFAULT-MASK MASK '**, **$.99'
  ROW 6 'Reverting a Mask ' FIELD-WITH-DEFINED-MASK NOMASK
  ROW 7 'BWZ Mask' FIELD-WITH-BWZ-MASK
    
```

Mask Examples	
Using Default Mask	1,234.56
Using Defined Mask	\$1,234.56
Applying a Mask	*1,234.56
Reverting a Mask	1,234.56
BWZ Mask	

Hexadecimal Mask Example

CA-Easytrieve allows you to display data in hexadecimal format. A hexadecimal mask can be applied to fields of any data type, including alphanumeric (except for varying length fields). This feature allows you to display the actual contents of a field in double-digit hexadecimal format. When used with a screen input field, you can use CA-Easytrieve to enter or modify data in hexadecimal format. CA-Easytrieve automatically checks each digit for validity (0 through F) and returns any errors for correction.

```

SCREEN NAME SCREEN1
  TITLE 1 'Mask Examples'
  ROW 3 'Name . . . .' EMPNAME MASK HEX
  ROW 5 'Gross Pay . .' GROSS-PAY MASK HEX
    
```

Mask Examples	
Name	C2C5D9C7404040404040404040404040404040404040404040
Gross Pay . .	0075920C

Automatic Editing of Input

CA-Easytrieve automatically edits input data with little or no coding required. CA-Easytrieve performs the following types of edits:

- Data type validation
- Upper casing
- Value checking
- Mask checking
- Pattern matching

The order in which CA-Easytrieve performs editing is:

1. If UPPERCASE is specified for the field, translate the field to all uppercase characters.
2. If a PATTERN is specified for the field, edit the data against the pattern. See PATTERN later in this chapter for details.
3. If a MASK is specified for the field, edit the data against the mask, including data type validation.
4. If a VALUE is specified for the field, edit the data against the value.

See Setting Errors later in this chapter for information on how you can use the SET statement to edit input data.

UPPERCASE

You can specify UPPERCASE for fields coded on a ROW statement. When UPPERCASE is coded, CA-Easytrieve converts data entered on the screen to uppercase characters as it is received from the terminal. To convert all fields on the screen to uppercase, code UPPERCASE on the SCREEN statement.

MASK

CA-Easytrieve automatically edits the data entered in a numeric field according to an edit mask (default or override)..

- Allow and accept digits, a leading or trailing sign (but not both), and a single decimal point. Leading signs are + or -. Trailing signs are +, -, or the trailing string in the mask (for example, CR). Leading and trailing blanks are accepted and discarded.
- Align the decimal point when the data is received from the screen.

For example, if you code the following field:

```
DEFINE NUMFLD W 3 P 2 MASK 'ZZ9.99'
```

data is actually placed into the field as follows:

If the user types...	Then CA-Easytrieve stores:	
1	001	.00
1.2	001	.20
.235	000	.24

CA-Easytrieve automatically rounds the data to fit the field.

Decimal alignment is performed only for input. When you display data with a mask, there is no implied relationship between the mask and the number of decimal digits in the field.

- Allow and discard characters that appear in the mask that are displayed along with the data. For example, commas appearing in a quantitative field or parentheses and a hyphen appearing in a telephone number are discarded from the input before the data is stored in the actual field.

Display characters in the data must occur in the same order they appear in the mask. The characters, however, are not required to appear in the data. For example, applying the mask, 99,999.99, against the data, 12345.67 does not cause an error condition. However, applying the mask, '(999) 999-9999', against the data, '(617 322-2762)' is in error because the display characters are out of order.

Any other characters not appearing in the mask cause an error message to be displayed to the terminal user. Blanks imbedded in the middle of data that are not part of the mask also cause an error condition.

- Allow only numeric data for numeric fields. A blank entry is allowed only when the mask specifies blank when zero. A field is blank when zero when the mask is BWZ or contains only Z's for digits.
- Fields that use a hexadecimal mask (MASK HEX) have the input data automatically validated for correct double-digit hexadecimal characters (0 through F). You can display numeric or alphanumeric (except VARYING) fields with the hexadecimal mask. If MASK HEX is applied to a numeric field, CA-Easytrieve only edits data for a valid hexadecimal format, not that data is numerically valid.

PATTERN

PATTERN allows each input character to be edited according to the pattern specified. A pattern is a sequence of characters that describe the format of the data in the field. You can use a PATTERN to edit complex combinations of data types and character sequences. (A MASK is used only for numeric data. If you use arithmetic on the data, you probably do not want to use a PATTERN.)

You typically use a PATTERN to edit a field that contains a mixture of alphabetic and numeric characters in a specific sequence.

For example, to specify a part identification that is a five-character alphanumeric field where the first and last characters must be uppercase alphabetic (U) and the middle three characters must be numeric digits (D), code:

```
ROW 'Part ID . . .' PART-ID PATTERN 'UDDDU'
```

This pattern tells CA-Easytrieve to accept entries such as A123C and reject entries such as 1A23C.

Valid PATTERN Characters

The valid pattern characters and their meanings are listed in the following table.

Character	Meaning
A	Represents an uppercase or a lowercase 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 the 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 between alternatives.
(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 within the 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 the character set in the pattern. This operation removes from the data the character that matched the character set.
+	Represents character set addition to form another character set.

Step 2: The pattern can best be described by:

- Accept 0 through 9 for position 1.
- Accept 0 through 9 for position 2.
- Accept 0 through 9 for position 3.
- Accept 0 through 9 for position 4.
- Accept 0 through 9 for position 5.

Step 3: The best pattern character for each position is D, since D represents the character set of digits. The pattern becomes: DDDDD.

Step 4: Because there is only 1 order expected for this field, Step 4 does not apply.

Step 5: Because D repeats 5 times, the pattern can be D(5). In this case, the application of this step is not required. CA-Easytrieve internally generates the same for DDDDD and D(5). You can use either pattern.

Building a Name Pattern

If you have a field that represents a first name:

Step 1: Analyze your requirements:

- All blanks are acceptable.
- If a name is keyed, the first character must be uppercase and the remaining characters must be lowercase.
- If a name is keyed, there can be no leading blanks.
- If a name is keyed, trailing blanks are acceptable.
- If only an initial is keyed, it must be uppercase and it must be followed by a period. The remainder of the field must be blank.
- If an initial is keyed, there can be no leading blanks.

Step 2: There are 3 possible orders for how the characters for this field can be keyed:

- The first order is all blanks (requirement 1).
- The second order is an uppercase character followed by one or more lowercase characters followed by 0 or more blanks (requirements 2, 3, and 4).
- The third order is an uppercase character followed by a period followed by blanks (requirements 5 and 6).

Steps 3, 4, and 5: The part of the pattern which corresponds to the first order is a repetition of B, since B represents blanks.

If the field is 10 characters long, one way to specify this order is B(10) or BBBBBBBBBB. A better way to specify this order is B*. B* means that an infinite number of blanks can be accepted. Since the field is only 10 bytes long, there can be at most 10 blanks to accept. The B* generates a much smaller internal representation, and also adapts better to changes in the size of the field.

The part of the pattern which corresponds to the second order is:

- A U for the uppercase character
- A repetition of L's for the lowercase characters
- A repetition of B's for the trailing blanks.

If the field is 10 characters long, there can be 1 through 9 lowercase letters. One way to specify the repetition is L(1..9), but a better way is to use L(1..*) since the length of the field enforces a practical limit. For the repetition of blanks, use B* instead of B(0..8). The part of the pattern for the second order is UL(1..*) B*. (Blanks imbedded in patterns are ignored.)

The part of the pattern which corresponds to the third order is:

- A U for the uppercase character
- A "." for the period
- A repetition of B's for the blanks.

Although there is always the same number of blanks, use B* to describe the trailing blanks in the order. B(*) produces a much smaller internal representation and is also more flexible. The part of the pattern for the third order is U "." B*.

Combining the parts into a single pattern results in:

```
'(B*) | (U L(1..*) B*) | (U "." B*)'
```

Character Sets

A pattern represents the order of character sets in which you accept the data from the screen for a particular field. The letters A, B, E, D, L, N, U, and X represent predefined character sets. A single letter enclosed in double quotes represents a character set consisting of one character. (A sequence of letters enclosed in double quotes represents a series of character sets.)

A character set indicates which characters are acceptable. For example, the letter U (when not enclosed in double quotes) indicates that any uppercase letter is acceptable.

Occasionally, you prefer to identify characters which do not fit into one of the predefined character sets. In these cases, you can build a character set that identifies exactly the characters you require. The + and the - operators enable you to add sets together or to obtain the set difference. The () enable you to control the precedence of the operations.

A frequent use for set difference is constructing a set of all characters except a specific set of characters. For example, a pattern to specify all characters except blanks is X-B.

A frequent use for set addition is constructing a set of characters consisting of one of the predefined sets plus a few additional characters. For example, a pattern to specify uppercase letters plus blanks is U+B.

Both U+B and U|B recognize the same data. The internal form of U+B is marginally better than U|B because U+B describes a character set; U|B does not. As a character set, the action operator # can be appended to the set. The set can be combined with another set using - or + to form a different set.

Note: If you need to specify a pattern for predefined character sets in another language, contact Computer Associates Technical Support.

Advanced Numeric Patterns

You can use a PATTERN to provide advanced editing of numeric data. For example, you can use a MASK to provide basic display and edit criteria for a nine-digit ZIP code:

```
ROW 13 'ZIP Code . . .' ZIP-CODE MASK '99999-9999'
```

The mask, when used alone, allows the user to simply type zero. To require the user to enter all nine digits with or without the hyphen, add the following PATTERN:

```
ROW 13 'ZIP Code . . .' ZIP-CODE MASK '99999-9999' +
      PATTERN 'D(5)"-"(0..1)D(4)B(*)'
```

The pattern specifies that there should be exactly five digits entered, followed by 0 to 1 hyphens followed by exactly four digits.

Following is a similar example of a PATTERN for a social security number:

```
ROW 13 'SSN . . .' SSN MASK '999-99-9999' +
      PATTERN 'DDD "-"(0..1) DD "-"(0..1) DDDDB(*)'
```

The PATTERN specifies that there should be exactly three digits entered, followed by 0 to 1 hyphens, followed by exactly two digits, followed by 0 to 1 hyphens, followed by exactly four digits, followed by any number of spaces.

Advanced Editing of Names

You can use a PATTERN to ensure the terminal operator enters valid data in a name field:

```
ROW 13 'Name . .' EMPNAME PATTERN 'U(1..*) B(*)'
```

The PATTERN specifies that only a single name of at least one uppercase-only character, followed by any number of spaces is valid.

To permit a single uppercase letter followed by only uppercase, or only lowercase letters followed by any number of spaces use:

```
ROW 13 'Name . .' EMPNAME PATTERN 'U(U(*),L(*) B(*)'
```

To strip leading blanks add:

```
ROW 13 'Name . .' EMPNAME PATTERN 'B#(*) U(U(*),L(*) B(*)'
```

To enable one or more names in uppercase with separation by only one space:

```
ROW 13 'Name . .' EMPNAME PATTERN 'U(1..*) (B U(1..*))(*) B(*)'
```

To permit uppercase or lowercase letters after the first uppercase letter:

```
ROW 13 'Name . .' EMPNAME PATTERN 'U(U(*),L(*) (B U(U(*),L(*)*) B(*)'
```

Miscellaneous Advanced Editing

This pattern strips leading blanks while not permitting an all blank field:

```
ROW 13 'Description . .' DESCRIPTION PATTERN 'B#(*) (X-B) X(*)'
```

X-B signifies that all characters except blanks are allowed. To permit an all blank field:

```
ROW 13 'Description . .' DESCRIPTION +
      PATTERN 'B(*) ((X-B)(1..*), B(*)*)'
```

To strip leading spaces and remove extra spaces:

```
ROW 13 'Description . .' DESCRIPTION +
      PATTERN 'B#(*) ((X-B)(1..*) B B#(*)*)'
```

Internal Operation of Patterns

This topic discusses some advanced material relating to the internal operation of patterns. This information is not required to build most patterns. However, if you have difficulty building a pattern with the # and (*m..n*) operators, this information is helpful.

Patterns are implemented as **state tables**. A state table examines each character in a string to determine if the string is acceptable or not. Each character examined causes a transition from one state to the next. Most compilers use state tables for recognizing tokens (for example, labels or identifiers).

In CA-Easytrieve, there are two extensions in the pattern language which are not always converted to a state table: the # and the (m..n) operators. These operations are implemented as actions which are associated with transitions.

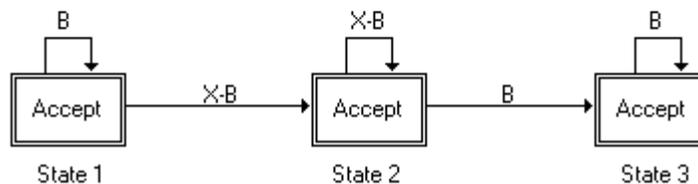
First Extension - TOSS Operator

The first extension is the toss operator, # or /-/. It is possible to compose a pattern which converts to a state table in which the transition from one state to the next does not know whether to toss or keep the recognized character. Consider the following pattern:

'B* (X-B)* B*'

This pattern allows data that consists of a single string of non-blank characters with or without leading and trailing blanks, or an all blank field. The following exhibit illustrates the state table and graphical representation of the table.

		Input character classes		
		B	X-B	end of input
C U R R E N T	State 1	State 1	State 2	accept
	State 2	State 3	State 2	accept
	State 3	State 3	error	accept



The body of the table indicates what CA-Easytrieve does when in a given state with given input. If the entry indicates a state, CA-Easytrieve goes to that state. If the entry indicates *accept*, the data is valid. If the entry is *error*, the data is rejected.

If you change the pattern to remove any leading blanks, the pattern becomes:

'B#* (X-B)* B*'

In the process of converting the pattern to the state table, all of the possible paths through the pattern are considered. If the first entry is a blank, CA-Easytrieve does not know whether to remove the character or not (does it match the B#* or the B*?). This pattern generates an error message when compiled.

To correct this pattern and achieve the same results, use:

'B#* (X-B) B#*'

This pattern removes leading and trailing blanks. When the first character is a blank, it does not matter if it is a leading or a trailing blank since both get removed. The loss of trailing blanks is not damaging. After the data has been accepted, CA-Easytrieve moves the data into the corresponding field in the library section. If the field is alphanumeric, it is padded with blanks. If it is a VARYING alphanumeric field, the length of the field reflects the data, less the trailing blanks. If the field is numeric, the conversion to the correct data format ignores the blanks.

Second Extension - Limited Repetition

The second extension is the range repetition with a fixed upper bound, (*m..n*). This type of repetition is called a limited repetition. The restrictions on its use are:

1. Limited repetitions cannot be nested within other repetitions (except (*m*)).
2. When a character from the input data is examined, there can be no ambiguity as to which is the next state and how to count the character.

Rule 1 is the result of the nature of state tables. All actions are done at the transition from one state to the next. The counting and the checking required by a limited repetition is done at a transition. It is impossible, in all cases, to assign an action to one or more transitions that reset the counters.

To illustrate Rule 2, consider the following pattern:

```
'D(0..2) (". "|E) D(0..2) B*'
```

If the first character in the input data is a digit, does it match the first D(0..2) or the second? If CA-Easytrieve could look ahead to see the rest of the input it could decide which one it was. But, as stated earlier, patterns examine only one character at a time. Therefore, this pattern generates an error message when compiled.

Note: The pattern 'D(0..2) (". "|E) D(0..2)' does not appear to have any practical application.

Additional Considerations

Consider the following pattern:

```
'B* X'
```

If the first character to be examined by the pattern is a blank, does the blank match the B* or the X? If it matches the B* there can be more data. If it matches the X, there can be no more input. Therefore, this pattern generates an error message when compiled.

When CA-Easytrieve gets data back from the screen, the data is almost always padded with trailing blanks unless you filled the entire field with data. The EOF key on the 3270 terminal may set the rest of the field on the screen to nulls, but by the time the pattern sees the data, the nulls have been converted to blanks. You may want to allow for trailing blanks in your patterns.

Note: Trailing blanks in displayed VARYING alphanumeric fields are converted to nulls when received.

VALUE

CA-Easytrieve automatically edits the input data against the value(s) specified in the VALUE parameter for the field on the ROW statement. You can automatically edit the data against a single value, a range of values, or a series of values. The VALUE parameter works similar to a CA-Easytrieve IF statement.

When CA-Easytrieve edits an alphanumeric field:

- The values must be alphanumeric literals enclosed in quotes.
- The comparison is based on the greater of the length of the value and the length of the field. The shorter item is padded with blanks out to the length of the longer item. This rule is subject to the exception below.
- When a fixed length field is compared with a longer fixed length value, the comparison is based on the length of the field. The value is truncated to match the length of the field. A warning message is generated by the compiler.
- The comparison is logical (bit-by-bit).

When CA-Easytrieve edits a numeric field:

- The values must be numeric literals.
- Comparison is arithmetic.

The following ROW statements illustrate automatic value editing:

```
DEFINE ALPHA-FIELD  W 1 A
DEFINE NUMERIC FIELD W 3 N 0
ROW 'Alpha Test . . .' ALPHA-FIELD VALUE ('A', 'D', 'U')
ROW 'Numeric Test . .' NUMERIC-FIELD VALUE (1, 101 THRU 500, 999)
```

Edit Error Messages

As a result of the automatic editing process, CA-Easytrieve handles error conditions as follows:

- Input fields are edited as found on the screen from left-to-right and top-to-bottom.

- All input fields are edited each time a user presses a programmable key (for example, PA keys, function keys, Enter) unless the key pressed is an IMMEDIATE key. IMMEDIATE keys cause the screen to be received but the data is not edited and moved into program fields. See Screen Key Processing, later in this chapter for details.

Note: If you are executing in the workstation environment, CA-Easytrieve performs much of the automatic editing as the data is entered in the field. It does not wait for a programmable key to be pressed before editing the data.

- Each field found in error receives the FIELD ERROR attribute. You can specify this attribute on a DEFAULT FIELD ERROR statement at the beginning of the screen declaration. If you do not use the DEFAULT FIELD ERROR statement, this attribute is taken from the site options. You can set individual error attributes for a field using the ERROR ATTR parameter for the field on the ROW statement.
- CA-Easytrieve automatically displays a message describing the error for the first field in error on the screen. This message typically tells the user that the field did not match one of the values permitted for the field or that the format of the data was incorrect. You can override the system-issued message with your own message using the ERROR parameter for the field on the ROW statement.

Messages issued for editing errors, whether system-issued or from the ERROR parameter, are always an ACTION message level. See Screen Message Area, later in this chapter for more information.

Cursor Placement

You can specify the placement of the cursor on the screen in the following ways:

- Use the CURSOR attribute in the ATTR parameter for the field on the ROW statement. You can also specify the CURSOR attribute for a field flagged in error using the ERROR ATTR parameter for the field on the ROW statement.
- Execute a CURSOR statement in a screen procedure to specify the field on the screen that receives the cursor upon the next display of the screen.

Cursor Placement Hierarchy

When more than one way is used to place the cursor in a specific location, CA-Easytrieve uses the following hierarchy to determine how the cursor is placed. The priority is from highest to lowest.

1. If the screen is re-displayed using a RESHOW action (see Screen Procedures later in this chapter), the cursor is placed in the same position as when the screen was received, regardless of any other method used.

2. If the field is detected in error by the automatic editing process or a SET statement, the first field containing the CURSOR attribute on the ERROR ATTR parameter of the ROW statement receives the cursor. *First* is defined as left-to-right, top-to-bottom.
3. If not a RESHOW or error condition, a CURSOR statement executed in a screen procedure names the field to receive the cursor. If the CURSOR statement is executed more than once before the screen is displayed, the last CURSOR statement executed determines cursor placement.
4. If a CURSOR statement is not executed, the first field on the screen with the CURSOR attribute receives the cursor.
5. If no field on the screen contains the CURSOR attribute, the first modifiable field on the screen receives the cursor.
6. If there are no modifiable fields on the screen, the first item on the screen receives the cursor.

Repeating Rows of Data

CA-Easytrieve supports the display of arrays on the screen with the REPEAT and END-REPEAT statements. You use the REPEAT and END-REPEAT statements to surround one or more ROW statements in the screen declaration, and specify the number of times that the enclosed group of ROW statements is repeated on the screen.

The REPEAT and END-REPEAT statements can also be used to name an CA-Easytrieve subscript field which is incremented each time the repeated rows are displayed. The subscript field helps automate the process of moving the array to the screen and then back again upon receiving the data when you use it for the array elements. You can define the subscript field or let CA-Easytrieve automatically define it for you.

A Simple REPEAT Example

Following is an example of how to display an array in a file on the screen. CA-Easytrieve automatically defines the subscript field, POLICY-SUB, used on the REPEAT statement.

```
FILE POLICIES INDEXED
CUST-NO      *    5  N
CUST-NAME   *   20  A
POLICY-NO   50   8  N   OCCURS 3. * Customer may have up to 3
                               . * policies

SCREEN NAME SHOW-POLICIES
TITLE 'View Customer Policy Numbers'
ROW 3 'Customer Number . .' CUST-NO
ROW 5 'Customer Name . . .' CUST-NAME
ROW 7 'Policies'
REPEAT 3 TIMES VARYING POLICY-SUB
  ROW POLICY-NO (POLICY-SUB)
END-REPEAT
```

```

View Customer Policy Numbers
Customer Number . . 10346
Customer Name . . . JONES

Policies
32894671
65274902
76642915
    
```

Two-dimensional Arrays

As illustrated in the previous example, you can code a subscript on each element of an array within a REPEAT group. If these fields are elements of a two-dimensional array, you can also add a second subscript. CA-Easytrieve does not automatically increment a second subscript for you. The second subscript must be specified as a literal or as a static field. The following example illustrates this:

```

...
WS-EMPLOYEE W 33 A OCCURS 3 . * 2-dimensional table of
WS-NAME WS-EMPLOYEE 30 A . * 3 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 3 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
    
```

```

List of Employees
Name                               Statuses
WIMN, GLORIA                       F G O
BERG, NANCY                         C
CORNING, GEORGE                     I T
    
```

Screen Message Area

The message area is used to display system and programmer-issued messages to the terminal user. CA-Easytrieve allows you to issue different levels of messages depending on the severity of the error. The three message levels are (in order of ascending severity):

- INFORMATION
- WARNING
- ACTION

Information messages typically inform a user that processing is proceeding normally. Warning messages tell the user that a potentially undesirable result has occurred or could occur. Action messages tell users that an action is required to correct a situation.

System-issued messages and messages specified on the SET statement are always message level ACTION.

Message Area Location

The default message area location is at the bottom of the screen, just above the function key display area. You can move the message area by specifying its row number on a DEFAULT MESSAGE statement at the beginning of the screen declaration. By default, all three levels of messages are sent to the same screen row number. Use the DEFAULT statement to designate from one to three rows to display different levels of messages on the screen.

Message Attributes

You can specify attributes for the three levels of messages on DEFAULT MESSAGE statements. If you do not use a DEFAULT MESSAGE statement, screen attributes for each message level are taken from the site options.

Message Text

The screen message area is used both for system-issued and programmer-issued messages. System-issued messages result from the edit process that CA-Easytrieve automatically performs on input data. You can override the message resulting from the edit process with the ERROR parameter of the ROW statement. See Automatic Editing of Input earlier in this chapter. You can also issue messages from the screen procedures that you code following the screen declaration by executing the MESSAGE or SET statement prior to the display of the screen.

Screen Function Key Area

The optional function key area is used to tell the terminal user which function keys are active and the action each performs.

Location

The function key area is always located at the bottom of the screen. The display is determined by the KEY statements you code in the screen declaration. You can specify descriptive text on each KEY statement to be displayed with the name of the key. Depending on the number of keys and the length of the descriptive text, more than one row of the screen may be required to display the active function keys.

Note: If you specify that one or more message areas use the same screen row as the function key area, messages may overlap the function key area. The default location for messages is just above the function key area.

Attributes

You can specify attributes for the function key area on a DEFAULT KEY statement. If you do not use a DEFAULT KEY statement, screen attributes for the function key area are taken from the site options.

Screen Key Processing

CA-Easytrieve automatically validates keys that the terminal user presses to send the screen to the program for processing. You control this process by coding KEY statements in the screen declaration. KEY statements determine which keys are valid on a particular screen.

Keys can also be assigned to perform specific actions when the key is pressed. As described earlier in Screen Function Key Area, you also control the display of information about the keys to the terminal user on KEY statements.

The following rules apply to key processing in CA-Easytrieve:

- Each KEY statement specifies one or more keys that are active for that SCREEN activity. If the user presses an inactive key, CA-Easytrieve automatically sends an error message to the user.
- Each KEY statement can specify a NAME parameter containing text to be displayed at the bottom of the screen.
- If you do not code any KEY statements in your SCREEN activity, all keys are active and you must provide code in your SCREEN procedures to validate key values.
- With each KEY statement, you can optionally assign the key(s) to automatically perform a branch action. Branch actions cause activity execution to branch to a specific step in the process. The branch actions you can code on a KEY statement and their effects are:

Action	Effect
REFRESH	Restore the initial condition of the screen
EXIT	Terminate the SCREEN activity

See Screen Procedures later in this chapter, for more information.

- With each KEY statement, you can optionally assign the key to perform IMMEDIATE processing. IMMEDIATE indicates that the key is to be processed immediately, without editing the input data and moving the data into the program fields.
- KEY statements can only be coded for keys for which CA-Easytrieve has defined a symbolic name. The symbolic names are assigned to specific values of the system-defined field, KEY-PRESSED. See System-defined Fields earlier in this chapter, for symbolic names.

Note: If you process values of KEY-PRESSED that do not have symbolic names (in screen procedures) you cannot code KEY statements in your screen declaration.

3270 Display Station Keys

When the terminal user presses CLEAR, PA1, PA2, or PA3, the 3270 Display Station returns only the name of the key pressed. Data on the screen is not received and the cursor position cannot be determined.

Screen Procedures

The execution of a SCREEN activity is actually a collection of procedures that CA-Easytrieve performs in a certain sequence. There are four points in a SCREEN activity in which CA-Easytrieve invokes special-named procedures. You can code these special-named procedures to perform customized actions specific to your application. The special-named screen procedures are:

- INITIATION
- BEFORE-SCREEN
- AFTER-SCREEN
- TERMINATION

You are not required to code these procedures. If not coded, CA-Easytrieve simply proceeds to the next step in the activity.

The following exhibit illustrates the SCREEN activity process that CA-Easytrieve performs and when the special-named procedures, if coded, are executed.

Step 1: Reset working storage then perform INITIATION procedure, processing any branch actions.

Step 2: Reset working storage then perform BEFORE-SCREEN procedure, processing any branch actions.

Step 3: Build the screen using program fields, pending messages, and pending cursor placement. Clear the internal pending message buffer.

Step 4: Send the screen to the terminal. Terminate and resume the program (if pseudo-conversational). Receive the screen from the terminal

Step 5: If KEY-PRESSED is an IMMEDIATE key, go to step 7.

Step 6: If KEY-PRESSED is not an IMMEDIATE key, edit input data. If any errors, go to step 4. If no errors, move the data into the program fields.

Step 7: If KEY-PRESSED has an associated branch action, perform it.

Step 8: Perform AFTER-SCREEN procedure, processing any branch actions.

Step 9: Go to Step 2.

Step 10: When EXIT is requested, reset working storage, then perform TERMINATION procedure, processing any branch actions.

INITIATION

The INITIATION procedure is performed one time during the initiation of the activity. Use INITIATION to perform actions that can only be executed once, for example, setting a field to an initial value or positioning a file at a specific starting location. Work fields with the RESET parameter specified are automatically initialized before the INITIATION procedure is invoked.

The REFRESH and RESHOW branch actions are invalid in an INITIATION procedure. See Branch Actions, later in this chapter, for details.

BEFORE-SCREEN

The BEFORE-SCREEN procedure is invoked during each iteration of the SCREEN activity. It precedes building the screen and the terminal I/O process. Typically, BEFORE-SCREEN is used to perform file I/O, initialize fields, or set the cursor position. Work fields with the RESET parameter specified are automatically initialized before the BEFORE-SCREEN procedure is invoked.

GOTO SCREEN, REFRESH, and RESHOW are invalid in a BEFORE-SCREEN procedure. See Branch Actions, later in this chapter, for details.

AFTER-SCREEN

The AFTER-SCREEN procedure is performed during each iteration of the activity after the terminal I/O processes. It is not executed if the key pressed is assigned to execute a branch action. An AFTER-SCREEN procedure can be used to perform complex editing and to update files with data entered on the screen.

All branch actions are valid in the AFTER-SCREEN procedure. See Branch Actions later in this chapter, for details.

TERMINATION

The TERMINATION procedure is performed when an EXIT action is executed, either from a key pressed or from another screen procedure. It is used to perform actions that are to be executed only at the end of the activity. Work fields with the RESET parameter specified are automatically initialized before the TERMINATION procedure is invoked.

If GOTO SCREEN or EXIT are executed in a TERMINATION procedure, the activity is terminated. REFRESH and RESHOW are invalid in a TERMINATION procedure. See Branch Actions later in this chapter, for details.

Programmer-Defined Procedures

You can code your own procedures in a SCREEN activity and perform them from the special-named screen procedures. Procedures you code are local to the screen activity and cannot be performed from other activities.

Branch Actions

There are four actions that cause the program execution to branch to specific steps in the SCREEN activity process:

Action	Step	Effect
GOTO SCREEN	2	Repeat the activity process.
REFRESH	3	Restore the initial condition of the screen.
RESHOW	4	Re-display the screen as it was received.
EXIT	10	Terminate the activity.

GOTO SCREEN

You can use the GOTO SCREEN statement to repeat the activity process. The default action of a SCREEN activity is to repeat the process until an EXIT action is executed. If the bottom of the process (the end of the AFTER-SCREEN procedure) is reached, the activity simply repeats, starting with the BEFORE-SCREEN procedure. (The INITIATION procedure is a one-time-only procedure).

You can code the GOTO SCREEN statement to cause an immediate branch to the top of the activity. This is similar to the way in which GOTO JOB branches to the top of a JOB activity.

REFRESH

REFRESH causes the screen to be restored to its initial condition or updated to reflect the current status of information. CA-Easytrieve rebuilds the screen using the current value of the fields specified on the screen.

When REFRESH is coded on an IMMEDIATE key, CA-Easytrieve ignores data entered on the screen and refreshes the screen just as it was originally sent to the user. When REFRESH is coded on a non-IMMEDIATE key, it causes data entered to be edited and moved into the program field areas, but no special-named procedure is invoked. This enables you to assign a key solely to allow the user to edit data. The data entered is edited against the automatic edits you code and no additional code is required in the AFTER-SCREEN procedure. When the user wants to actually update the file with the data, he could press another key to invoke the AFTER-SCREEN procedure. This is illustrated in the following example.

```
SCREEN NAME MYSCREEN
  TITLE 'Inventory Control'
  KEY F5 NAME 'Refresh' IMMEDIATE REFRESH
  KEY F6 NAME 'Edit input' REFRESH
  KEY F10 NAME 'Update'
  ...
```

When the user presses F5, data currently on the screen is ignored and the screen is rebuilt from the original field contents. When the user presses F6, the data entered is edited as specified on the ROW statements. Pressing F10 also edits the data but then performs an AFTER-SCREEN procedure that updates the file.

You can also use REFRESH to update the screen contents with the current data available. For example, a user enters a quantity and a price and wants to see the extended price (price times quantity). You can use a REFRESH coded in an AFTER-SCREEN procedure to compute the extended price as shown in the next example.

```
SCREEN NAME MYSCREEN
  TITLE 'Inventory Control'
  KEY F2 NAME 'Reset to zero'
  KEY F6 NAME 'Refresh' IMMEDIATE REFRESH
  KEY F9 NAME 'Show Extended Price'
  KEY F10 NAME 'Update'
  ROW 3 'Quantity . .' QUANTITY
  ROW 5 'Price . . . .' PRICE
  ROW 7 'Ext Price . .' EXT-PRICE ATTR (TURQ ASKIP)
  ...
  BEFORE-SCREEN. PROC
    MOVE ZEROS TO QUANTITY, PRICE, EXT-PRICE
  END-PROC
  AFTER-SCREEN. PROC
    IF KEY-PRESSED = F9
      EXT-PRICE = PRICE * QUANTITY
      REFRESH
    END-IF
  ...
END-PROC
```

When the user types the quantity and price and presses F9, the quantity and price are received, and the extended price is computed. The REFRESH then re-displays the screen using the current value of the program fields, and the newly-computed extended price is displayed.

RESHOW

RESHOW can be used in an AFTER-SCREEN procedure to re-display the screen after the screen has been received. CA-Easytrieve saves a copy of the screen image it receives. You can then EXECUTE another SCREEN activity. When the program returns to the first activity, use RESHOW to re-display the saved image of the first screen.

When associated indirectly with an IMMEDIATE key, you can ignore any data entered on the screen, display a second screen, then RESHOW the first screen intact. For example, you can permit the user to view a help screen, then return to the screen on which the user requested help. See Providing Help Screens later in this chapter, for an example of program code.

When associated indirectly with a non-IMMEDIATE key, you can permit the user to display a selection list, accept and process the user's selection(s), then re-display the original screen.

EXIT

EXIT terminates the SCREEN activity and returns control to the activity from which it was EXECUTEd. If the current SCREEN activity was not EXECUTEd from another activity, EXIT terminates the program. Associating EXIT with an IMMEDIATE key is equivalent to a cancel function. Any data on the screen is ignored and the activity terminates. Associating EXIT with a non-IMMEDIATE key saves the data into the program fields after editing it.

Note: It is your responsibility to save the data to a file if your application requires it. Data saved into the program fields is lost when the program terminates unless written to a file.

CICS Pseudo-conversational Programs

The SCREEN activity process illustrated in the ten steps under Screen Procedures earlier in this chapter shows how CA-Easytrieve handles pseudo-conversational programs. In Step 4, CA-Easytrieve sends the screen to the terminal. If your SCREEN activity is executing in a pseudo-conversational mode (COMMIT NOTERMINAL is not specified for a CICS program), CA-Easytrieve then terminates the task and returns to CICS. When the terminal user presses a programmable terminal key, CICS executes the program again, and CA-Easytrieve resumes the task and receives the screen. See Commit Processing later in this chapter, for more information.

Also see Units of Work/Commit Processing and Coding Programs That Run Under CICS in “Coding a CA-Easytrieve Program” chapter for additional information.

Sending Messages

CA-Easytrieve maintains an internal message area for each message area defined on your screen. The MESSAGE statement can be executed anywhere in your program to update the pending message area. When the next screen is displayed, the screen message area is built from the pending message. The pending message area is then cleared.

You can use these pending messages across activities to prepare messages in one activity for display on a screen in another activity. The following example shows how to code messages in a SCREEN activity, and show the resulting screen.

```
FILE PERSNL INDEXED WORKAREA 150
%PERSNL
SCREEN NAME VIEW-UTILITY
KEY ENTER
KEY F3 NAME 'Exit' EXIT
TITLE 1 'Employee View Utility'
ROW 3 'Employee Number . .' EMP#
ROW 5 'Employee Name . . .' EMPNAME
INITIATION. PROC
```

```

MESSAGE 'Enter an employee number.' LEVEL INFORMATION
MOVE ZERO TO EMP-NO
MOVE SPACES TO EMPNAME
END-PROC
AFTER-SCREEN. PROC
READ PERSNL KEY EMP# STATUS
IF NOT PERSNL
MESSAGE 'Employee not found. Enter another number.' LEVEL ACTION
ELSE
MESSAGE 'Employee found. Enter another number.' LEVEL INFORMATION
END-IF
END-PROC

```

Employee View Utility	
Employee Number . . .	_
Employee Name . . .	
...	
Enter an employee number.	
F3=Exit	

An introductory message (message level INFORMATION) is issued to the terminal user from an INITIATION procedure. After the user enters an employee number and presses the Enter key, the AFTER-SCREEN procedure is performed and issues an appropriate message describing the results of the file I/O. If the record is found, the user receives an INFORMATIONal message. If not found, an ACTION message is issued, flagging an error condition that must be corrected before the process can continue.

Using Message Levels

CA-Easytrieve allows you to automatically manage different levels of messages. For example, you can issue INFORMATION or WARNING messages in your program, but discover later in the process that a severe error condition that requires an ACTION message could arise. If the message areas for the different message levels are located on the same screen row, coding the ACTION message overlays any previous or subsequent INFORMATION or WARNING messages because of the severity hierarchy of the messages levels. When you issue multiple messages of the same level, the last one issued for the highest level is displayed.

Determining the Cursor Location

You can determine the position of the cursor when the screen is received by using the special IF CURSOR statement in a screen procedure. You use the IF CURSOR statement to test whether the cursor is present within a specified field. The following example illustrates the IF CURSOR statement testing for cursor placement on any of four subscripted array elements on the screen.

```

DEFINE OPTION W 1 A OCCURS 4
SCREEN NAME MENU
KEY F2 NAME 'Select'
KEY F3 NAME 'Exit' EXIT
TITLE 1 'Employee System Main Menu'
ROW 3 'Position the cursor by your selection, press F2 to select.'
ROW 5 COL 10 OPTION (1) 'View employee'
ROW COL 10 OPTION (2) 'Edit employee'
ROW COL 10 OPTION (3) 'Delete employee'
ROW COL 10 OPTION (4) 'Add employee'
AFTER-SCREEN. PROC
  IF OPTION (1) CURSOR
    EXECUTE VIEW-EMPLOYEE
  ELSE-IF OPTION (2) CURSOR
    EXECUTE EDIT-EMPLOYEE
  ELSE-IF OPTION (3) CURSOR
    EXECUTE DELETE-EMPLOYEE
  ELSE-IF OPTION (4) CURSOR
    EXECUTE ADD-EMPLOYEE
  ELSE
    MESSAGE 'Position cursor by a menu selection.'
  END-IF
END-PROC
...

```

```

Employee System Main Menu

Position the cursor by your selection, press F2 to select.

  _ View employee
  _ Edit employee
  _ Delete employee
  _ Add employee

...
F2=Select  F3=Exit

```

Testing for Field Modification

You can use the IF MODIFIED statement to test whether a field was modified by the terminal user. Following are the rules for using the IF MODIFIED statement:

- The test for modification determines whether the value of the field actually changed. If the user types the same value in the field as was originally displayed, the modification test is false.
- The results of the IF MODIFIED test are set at the time the screen is received. If the value of the input data is not equal to the value of the program field, the field was modified. If the input data is equal to the program field, the field is considered not modified.
- The IF MODIFIED comparison is performed logically for both alphanumeric fields and numeric fields.

- If the screen is received as the result of an IMMEDIATE key, PA key, or CLEAR key, the IF MODIFIED test is always false.

Using the IF MODIFIED test can help you write more efficient programs. For example, you may not want to perform complex editing on a field unless it was changed by the user. The following is an example of the IF MODIFIED test.

```
SCREEN NAME EDIT-EMPLOYEE
KEY ENTER
KEY F3 NAME 'Exit' EXIT
TITLE 1 'Employee Edit Utility'
ROW 3 'Enter the employee number and new job category.'
ROW 5 'Employee number . .' EMP#
ROW 7 'Job Category . . . .' JOB-CATEGORY
AFTER-SCREEN. PROC
  IF JOB-CATEGORY MODIFIED
    PERFORM SEARCH-JOB-CATEGORY-TABLE
    IF NOT JOBTABLE
      MESSAGE 'Job category is invalid. Please reenter.'
    ELSE
      PERFORM UPDATE-EMPLOYEE
      MESSAGE 'Job category updated.' LEVEL INFORMATION
      MOVE ZERO TO EMP# JOB-CATEGORY
    END-IF
  ELSE
    MESSAGE 'Job category not modified.' LEVEL WARNING
    JOB-CATEGORY = 0
  END-IF
END-PROC
...
```

The IF MODIFIED test of the JOB-CATEGORY field determines whether or not to look up the field in a table and to update the record. If the user does not modify the screen contents, these I/O operations are not performed.

Setting Errors

When you can determine the validity of user input with a simple IF statement, such as IF DEPT = 901 THRU 999, the VALUE parameter on the ROW statement provides easy automatic error handling. Often, however, you must provide more complex logic to determine the validity of a value. For example, you may need to perform cross-field editing where the value of one field on the screen determines acceptable values for another field on the screen, or you may have to perform a table look-up or other advanced processing to determine validity.

You can use the SET statement to extend automatic error handling in screen procedures. When you execute a SET statement for a field, CA-Easytrieve uses the SET information the next time the screen is displayed. You can simply change the field's screen attributes; however, if you code the ERROR parameter, the screen process treats the field as if automatic editing detected the error. The field's error attributes and error message are used by default but you can even override these with a SET statement. See Sample Screen Applications later in this chapter, and the SET Statement in the *CA-Easytrieve Language Reference Guide* for more information.

Commit Processing

You can use CA-Easytrieve commit processing in screen activities to provide file integrity during update operations.

See the following topics in this guide for additional information on how CA-Easytrieve performs commit processing:

- **Controlling Program Flow: Units of Work/Commit Processing** in the “Coding a CA-Easytrieve Program” chapter for information on instructing CA-Easytrieve to use either automatic or controlled commit processing during program execution.
- **Overview: Hold/Release Processing** in the “File Processing” chapter for how CA-Easytrieve processes requests to update records during file processing.

SCREEN COMMIT Parameter

The COMMIT parameter of the SCREEN statement controls the way CA-Easytrieve automatically issues commit points during screen processing. The TERMINAL | NOTERMINAL subparameter controls whether records are held during terminal I/O operations. In multi-user environments such as CICS or workstation networks, it is important to fully understand the use of TERMINAL | NOTERMINAL.

COMMIT TERMINAL tells CA-Easytrieve to commit during each terminal I/O. A commit point releases any holds you have active. Before the program can update a record, it must re-read the record after the user returns the screen to the program for processing. A user cannot inadvertently lock a record from being accessed by other users as he contemplates updating a record. COMMIT TERMINAL is the default for CA-Easytrieve screen activities. In a CICS environment, this is called *running the program pseudo-conversationally*.

COMMIT NOTERMINAL tells CA-Easytrieve not to commit during terminal I/O. In CICS, this is called *running conversationally*. The advantage of running conversationally is that the program needs to read a record only once, display it on the screen, then simply update it when instructed by the terminal user. The disadvantage is that the record is held until the user presses an attention key, the update is performed, or a commit point is explicitly issued.

Conversational Processing Example

The following example program illustrates using COMMIT NOTERMINAL to keep CA-Easytrieve from issuing a commit point during terminal I/O. The terminal user types an employee number then presses F5 to find the employee's record. When he modifies the social security number and presses F6, the employee's record is updated. Because UPDATE is coded on the FILE statement, the READ statement automatically holds the record until it is updated. This hold is in effect the entire time the user contemplates the change. The hold prohibits other users in a multi-user environment from accessing the record during this time. Not issuing a commit point also keeps the system from freeing system resources while the program is running.

```

FILE KPERSONL INDEXED UPDATE
%PERSONL
WORK-EMP# W 5 N
SCREEN NAME UPDATE-KPERSONL COMMIT NOTERMINAL
KEY F3 NAME 'Exit' EXIT
KEY F5 NAME 'Find employee'
KEY F6 NAME 'Update employee'
TITLE 'Update Social Security Number'
ROW 5 'Employee Number. . . . . ' WORK-EMP#
ROW 7 'Social Security Number . .' SSN
INITIATION. PROC
  SSN = 0
  MESSAGE 'ENTER EMPLOYEE NUMBER. PRESS F5 TO FIND.' +
    LEVEL INFORMATION
END-PROC
AFTER-SCREEN. PROC
CASE KEY-PRESSED
  WHEN F5
    READ KPERSONL KEY WORK-EMP# STATUS
    IF NOT KPERSONL
      MESSAGE 'EMPLOYEE NOT FOUND. ENTER NEXT EMPLOYEE.'
      MOVE ZERO TO SSN
    ELSE
      MESSAGE 'ENTER NEW SSN. PRESS F6.' LEVEL INFORMATION
      CURSOR AT SSN
    END-IF
  WHEN F6
    WRITE KPERSONL UPDATE STATUS
    IF FILE-STATUS NE 0
      MESSAGE 'EMPLOYEE NOT UPDATED. REASON=' FILE-STATUS
    ELSE
      MESSAGE 'SSN UPDATED. ENTER NEXT EMPLOYEE. PRESS F5.' +
        LEVEL INFORMATION
    END-IF
END-CASE
END-PROC

```

Pseudo-Conversational Processing Example

The next example shows the same program using COMMIT TERMINAL to tell CA-Easytrieve to issue a commit point during terminal I/O. COMMIT TERMINAL is the default if not specified. The commit point issued between the time the user sees the record and finally updates it allows other users to access the record. It also frees valuable system resources each time a terminal operation is performed.

The program must read the record again following the user's request to update it (F6). Any hold issued during the read for the find request (F5) is lost when the employee record is displayed on the terminal. The program specifically states NOHOLD on the READ statement for finding the record and HOLD on the READ statement for updating the record. If NOHOLD had not been specified, a hold would have been issued but released when the screen was displayed.

Re-reading the record presents an additional complexity. If the actual record is used as the screen data area (as in this example), the second READ statement causes the data entered by the terminal user to be lost. To handle this condition, simply define a work area to hold the record contents during the read operation. Restore the contents before the update takes place.

```
FILE KPERSNL INDEXED UPDATE
%PERSNL
WORK-EMP# W 5 N
WORK-AREA W 150 A
SCREEN NAME UPDATE-KPERSNL COMMIT TERMINAL
KEY F3 NAME 'Exit' EXIT
KEY F5 NAME 'Find employee'
KEY F6 NAME 'Update employee'
TITLE 'Update Social Security Number'
ROW 5 'Employee Number. . . . . ' WORK-EMP#
ROW 7 'Social Security Number . .' SSN
INITIATION. PROC
  SSN = 0
  MESSAGE 'ENTER EMPLOYEE NUMBER. PRESS F5 TO FIND.' +
    LEVEL INFORMATION
END-PROC
AFTER-SCREEN. PROC
CASE KEY-PRESSED
  WHEN F5
    READ KPERSNL KEY WORK-EMP# NOHOLD STATUS
    IF NOT KPERSNL
      MESSAGE 'EMPLOYEE NOT FOUND. ENTER NEXT EMPLOYEE.'
      MOVE ZERO TO SSN
    ELSE
      MESSAGE 'ENTER NEW SSN. PRESS F6.' LEVEL INFORMATION
      CURSOR AT SSN
    END-IF
  WHEN F6
    MOVE KPERSNL TO WORK-AREA
    READ KPERSNL KEY WORK-EMP# HOLD
    MOVE WORK-AREA TO KPERSNL
    WRITE KPERSNL UPDATE STATUS
    IF FILE-STATUS NE 0
      MESSAGE 'EMPLOYEE NOT UPDATED. REASON=' FILE-STATUS
    ELSE
      MESSAGE 'SSN UPDATED. ENTER NEXT EMPLOYEE. PRESS F5.' +
        LEVEL INFORMATION
    END-IF
END-CASE
END-PROC
```

Concurrent Updates

When many users concurrently update the same file in a pseudo-conversational environment, your program must handle certain conditions. During the time the terminal user is contemplating modifying a record, the record could be deleted or updated by another user. For example, when a user is shown the quantity of an item in inventory, by the time he deducts his order's quantity, another user may have depleted the stock. If you do not re-evaluate the quantity when you re-read the record for update, the user could sell stock he does not really have.

There are several coding conventions to handle these conditions. Two of the more common are:

- Set a flag in the record that is under consideration for update. All programs accessing the file must test the flag before allowing any update access.
- Save a copy of the data that is displayed to the user. Before updating it, compare the current data to the saved copy. If any changes are detected, display the current data and warn the user that there was an intermittent change to the record.

The next example contains the previous program in which code has been added to handle intermittent updates. A copy of the social security number displayed to the user is saved in the working storage field, SAVE-SSN. Code has been added to detect when the record is deleted by another user. When the read is successful, the social security number in the record is compared to that originally displayed to the user. If it is not the same, the user is warned that another user already changed the number, and he is asked to process the change again.

```
FILE KPERSNL INDEXED UPDATE
%KPERSNL
WORK-EMP# W 5 N
WORK-AREA W 150 A
SAVE-SSN W SSN
SCREEN NAME UPDATE-KPERSNL COMMIT TERMINAL
KEY F3 NAME 'Exit' EXIT
KEY F5 NAME 'Find Employee'
KEY F6 NAME 'Update employee'
TITLE 'Update Social Security Number'
ROW 5 'Employee Number. . . . .' WORK-EMP#
ROW 7 'Social Security Number . .' SSN
INITIATION. PROC
  SSN = 0
  MESSAGE 'ENTER EMPLOYEE NUMBER. PRESS F5 TO FIND.' +
    LEVEL INFORMATION
END-PROC
AFTER-SCREEN. PROC
  CASE KEY-PRESSED
    WHEN F5
      READ KPERSNL KEY WORK-EMP# NOHOLD STATUS
      IF NOT KPERSNL
        MESSAGE 'EMPLOYEE NOT FOUND. ENTER NEXT EMPLOYEE.'
        MOVE ZERO TO SSN
      ELSE
        SAVE-SSN = SSN
        MESSAGE 'ENTER NEW SSN. PRESS F6.' LEVEL INFORMATION
        CURSOR AT SSN
      END-IF
    WHEN F6
      MOVE KPERSNL TO WORK-AREA
      READ KPERSNL KEY WORK-EMP# HOLD STATUS
```

```

IF NOT KPERSNL
  MESSAGE 'EMPLOYEE ' WORK-EMP# ' NO LONGER ON FILE. +
    ENTER NEW EMPLOYEE.  PRESS F5.'
  MOVE ZERO TO SSN
  GOTO SCREEN
ELSE-IF SSN NE SAVE-SSN
  MESSAGE 'INTERMITTENT CHANGE DETECTED! RETYPE NEW VALUE +
    AND PRESS F6 TO UPDATE.' LEVEL WARNING
  SAVE-SSN = SSN
  GOTO SCREEN
END-IF
MOVE WORK-AREA TO KPERSNL
WRITE KPERSNL UPDATE STATUS
IF FILE-STATUS NE 0
  MESSAGE 'EMPLOYEE NOT UPDATED. REASON=' FILE-STATUS
ELSE
  MESSAGE 'SSN UPDATED. ENTER NEXT EMPLOYEE.  PRESS F5.' +
    LEVEL INFORMATION
END-IF
END-CASE
END-PROC

```

SQL Processing Example

The programs in the previous examples process an indexed file. This next example illustrates how the previous program (Concurrent Updates) looks when an SQL table is processed instead.

```

FILE KPERSNL SQL( SSRPRS0.PERSNL) UPDATE
  SQL INCLUDE ( EMP#, SSN) FROM SSRPRS0.PERSNL LOCATION *
MASK-SSN SSN SSN MASK('999-99-9999')
WORK-EMP# W EMP# MASK('99999')
WORK-AREA W 8 A
SAVE-SSN W SSN
SCREEN NAME UPDATE-PERSNL COMMIT TERMINAL
KEY F3 NAME 'Exit' EXIT
KEY F5 NAME 'Find Employee'
KEY F6 NAME 'Update employee'
TITLE 'Update Social Security Number'
ROW 5 'Employee Number. . . . . ' WORK-EMP#
ROW 7 'Social Security Number . .' MASK-SSN
INITIATION. PROC
  MASK-SSN = 0
  MESSAGE 'ENTER EMPLOYEE NUMBER. PRESS F5 TO FIND.' +
    LEVEL INFORMATION
END-PROC
AFTER-SCREEN. PROC
CASE KEY-PRESSED
  WHEN F5
    SELECT FROM KPERSNL +
      WHERE EMP# = :WORK-EMP#
    FETCH FROM KPERSNL
    IF NOT KPERSNL
      MESSAGE 'EMPLOYEE NOT FOUND. ENTER NEXT EMPLOYEE.'
      MOVE ZERO TO MASK-SSN
    ELSE
      SAVE-SSN = SSN
      MESSAGE 'ENTER NEW SSN. PRESS F6.' LEVEL INFORMATION
      CURSOR AT MASK-SSN
    END-IF
  WHEN F6
    MOVE KPERSNL TO WORK-AREA
    SELECT FROM KPERSNL +
      WHERE EMP# = :WORK-EMP# +
      FOR UPDATE
    FETCH FROM KPERSNL
    IF NOT KPERSNL
      MESSAGE 'EMPLOYEE ' WORK-EMP# ' NO LONGER ON FILE. +
        ENTER NEW EMPLOYEE.  PRESS F5.'
      MOVE ZERO TO MASK-SSN

```

```
        GOTO SCREEN
ELSE-IF SSN NE SAVE-SSN
    MESSAGE 'INTERMITTENT CHANGE DETECTED! RETYPE NEW +
           VALUE AND PRESS F6 TO UPDATE.' LEVEL WARNING
    SAVE-SSN=SNN
    GOTO SCREEN
END-IF
MOVE WORK-AREA TO KPERSNL
UPDATE KPERSNL
IF FILE-STATUS NE 0
    MESSAGE 'EMPLOYEE NOT UPDATED. REASON=' FILE-STATUS
ELSE
    MESSAGE 'SSN UPDATED. ENTER NEXT EMPLOYEE. PRESS F5.' +
           LEVEL INFORMATION
END-IF
END-CASE
END-PROC
```

Sample Screen Applications

Following are coding examples for several common screen applications. Included application examples are:

- Editing data and setting errors
- Using dynamic screen attributes
- Invoking other screen activities from a menu
- Providing help screens for the terminal user
- Providing field-specific help
- Windowed screens
- Action bar pull-downs.

Editing Data and Setting Errors

CA-Easytrieve ROW statements provide automatic editing and error handling. However, complex edits may require logic in the screen procedures. The next example illustrates using an instream table file to verify the value typed for an employee's job category. When the search fails, the SET statement is used to indicate the field as being in error and an appropriate error message is issued.

```
FILE KPERSNL INDEXED UPDATE WORKAREA 150
%PERSNL
FILE JOBTABLE TABLE INSTREAM
ARG 1 2 N
DESC 3 1 A
10X
25X
30X
...
ENDTABLE
SCREEN NAME EDIT-EMPLOYEE LINESIZE 80 ROWCOUNT 24
KEY ENTER
KEY F3 NAME 'Exit' EXIT
TITLE 1 'Employee Edit Utility'
ROW 3 'Enter the employee number and new job category.'
ROW 5 'Employee number . .' EMP#
ROW 7 'Job Category . . . .' JOB-CATEGORY
...
AFTER-SCREEN. PROC
IF JOB-CATEGORY MODIFIED
PERFORM SEARCH-JOB-CATEGORY-TABLE
IF NOT JOBTABLE
SET JOB-CATEGORY ERROR +
'Job category is invalid. Please reenter.'
ELSE
PERFORM UPDATE-EMPLOYEE
MESSAGE 'Job category updated.' LEVEL INFORMATION
MOVE ZERO TO EMP# JOB-CATEGORY
END-IF
ELSE
MESSAGE 'Job category not modified.' LEVEL WARNING
JOB-CATEGORY = 0
END-IF
END-PROC
SEARCH-JOB-CATEGORY-TABLE. PROC
DEFINE WCAT W 1 A
SEARCH JOBTABLE WITH JOB-CATEGORY GIVING WCAT
END-PROC
...
```

Using Dynamic Screen Attributes

The following example illustrates a screen activity that uses dynamic screen attributes. DECLARED attributes are used for the job category prompt text and JOB-CATEGORY field. These DECLARED attributes are initialized when created with the DECLARE statement. When an error occurs, the attributes of both the prompt text and field are changed to highlight the condition. When an error condition does not exist, the normal attributes are used.

```
DECLARE TEXT-ATTR ATTR (GREEN ASKIP)
DECLARE FIELD-ATTR ATTR (TURQ)
DECLARE NORMAL-TEXT-ATTR ATTR (GREEN ASKIP)
DECLARE ERROR-TEXT-ATTR ATTR (YELLOW INTENSE REVERSE ASKIP ALARM)
DECLARE NORMAL-FIELD-ATTR ATTR (TURQ)
DECLARE ERROR-FIELD-ATTR ATTR (YELLOW INTENSE REVERSE ALARM)
SCREEN NAME EDIT-EMPLOYEE
KEY ENTER
```

```

KEY F3 NAME 'Exit' EXIT
TITLE 1 'Employee Edit Utility'
ROW 3 'Enter the employee number and new job category.'
ROW 5 'Employee number . .' EMP#
ROW 7 'Job Category . . .' ATTR TEXT-ATTR +
      JOB-CATEGORY ATTR FIELD-ATTR
INITIATION. PROC
  EMP# = 0
  JOB-CATEGORY = 0
END-PROC
AFTER-SCREEN. PROC
  IF JOB-CATEGORY MODIFIED
    PERFORM SEARCH-JOB-CATEGORY-TABLE
    IF NOT JOBTABLE
      MESSAGE 'Job category is invalid. Please reenter.'
      TEXT-ATTR = ERROR-TEXT-ATTR
      FIELD-ATTR = ERROR-FIELD-ATTR
    ELSE
      PERFORM UPDATE-EMPLOYEE
      MESSAGE 'Job category updated.' LEVEL INFORMATION
      MOVE ZERO TO EMP# JOB-CATEGORY
      TEXT-ATTR = NORMAL-TEXT-ATTR
      FIELD-ATTR = NORMAL-FIELD-ATTR
    END-IF
  ELSE
    MESSAGE 'Job category not modified.' LEVEL WARNING
    JOB-CATEGORY = 0
    TEXT-ATTR = NORMAL-TEXT-ATTR
    FIELD-ATTR = NORMAL-FIELD-ATTR
  END-IF
END-PROC
...

```

Using a Menu

CA-Easytrieve allows you to implement a menu application by coding multiple SCREEN activities within one CA-Easytrieve program. This example shows how you can create a menu that executes other SCREEN activities as child screens. When the child screen terminates due to an EXIT, the parent screen executes until the next EXIT. This example also demonstrates the ability for the child screen to send a message to be displayed on the parent screen.

```

FILE PERSNL INDEXED UPDATE WORKAREA 150
%PERSNL
DEFINE OPTION W 1 A
SCREEN NAME MENU UPPERCASE
KEY ENTER
KEY F3 NAME 'Exit' EXIT
TITLE 1 'Employee System Main Menu'
ROW 3 'Type an option and an employee number, then press Enter.'
ROW 5 'Option . . . . . ' OPTION VALUE ('V', 'E', 'X') +
      ERROR 'Please type V, E, or X.'
ROW 7 COL 25 'V View employee'
ROW COL 25 'E Edit employee'
ROW 11 'Employee Number . .' EMP# ATTR MUSTENTER +
      ERROR 'Please enter an employee name.'
INITIATION. PROC
  EMP# = 0
END-PROC
AFTER-SCREEN. PROC
  CASE OPTION
    WHEN 'V'
      EXECUTE VIEW-EMPLOYEE
    WHEN 'E'
      EXECUTE EDIT-EMPLOYEE
  END-CASE
END-PROC

SCREEN NAME VIEW-EMPLOYEE
DEFAULT FIELD ATTR (TURQ PROTECT)
KEY F3 NAME 'Exit' EXIT
TITLE 'Employee View Utility'
ROW 3 'Number . .' EMP#
ROW 'Name . .' EMPNAME
ROW 'SSN . .' SSN
ROW 'Dept . .' DEPT
ROW 'Phone . .' TELEPHONE
BEFORE-SCREEN. PROC
  READ PERSNL KEY EMP# STATUS
  IF NOT PERSNL
    MESSAGE 'Employee number not found. Please re-enter.'
  EXIT
END-IF
END-PROC

SCREEN NAME EDIT-EMPLOYEE COMMIT NOTERMINAL
KEY ENTER
KEY F3 NAME 'Exit' EXIT
TITLE 'Employee Edit Utility'
ROW 3 'Number . .' EMP#
ROW 'Name . .' EMPNAME
ROW 'SSN . .' SSN
ROW 'Dept . .' DEPT
ROW 'Phone . .' TELEPHONE
BEFORE-SCREEN. PROC
  READ PERSNL KEY EMP# STATUS
  IF NOT PERSNL
    MESSAGE 'Employee number not found. Please re-enter.'
  EXIT
END-IF
END-PROC
AFTER-SCREEN. PROC
  WRITE PERSNL UPDATE
  MESSAGE 'Employee updated successfully.' LEVEL INFORMATION
END-PROC

```

Providing Help Screens

CA-Easytrieve allows you to provide help screens for your terminal users. The example in this topic illustrates how you can display a help screen from the main menu used in the previous example.

The requirements to add a help screen to the menu are:

- Add a KEY statement to the menu screen for F1. F1 is designated as an IMMEDIATE key. This allows the user to request help even though the data on the screen is in error. If IMMEDIATE is not specified, the user continues to receive editing errors when he presses F1. He could never display the help screen for the help needed to continue with the application.
- In the AFTER-SCREEN procedure, test for when F1 is pressed. When true, EXECUTE the screen activity, MENU-HELP.
- After the EXECUTE statement, code a RESHOW statement. RESHOW tells CA-Easytrieve to re-display the contents of the screen as it was received when F1 was pressed. If the terminal user had entered data on the screen then requested help, RESHOW allows your program to display the help screen then re-display the original screen after the user exits the help screen. If REFRESH or GOTO SCREEN are used here, the user's data is lost.
- Add the MENU-HELP SCREEN activity. Since the KEY statement for F3 automatically exits the screen and no other processing is required, you do not need to code any screen procedures.

```

...
SCREEN NAME MENU UPPERCASE
KEY ENTER
KEY F1 NAME 'Help' IMMEDIATE
KEY F3 NAME 'Exit' EXIT
TITLE 1 'Employee System Main Menu'
ROW 3 'Type an option and an employee number, then press Enter.'
ROW 5 'Option . . . . . : OPTION VALUE ('V', 'E', 'X') +
      ERROR 'Please type V, E, or X.'
ROW 7 COL 25 'V View employee'
ROW COL 25 'E Edit employee'
ROW 11 'Employee Number . . ' EMP# ATTR MUSTENTER +
      ERROR 'Please enter an employee name.'
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 'Employee Menu Help'
ROW 3 COL 10 'The Employee System Menu provides the ability to view'
ROW COL 10 'or edit an employee's name, social security number,'
ROW COL 10 'department, or telephone number.'
ROW 7 COL 10 'Type V to view the employee data.'
ROW 9 COL 10 'Type E to edit the employee data.'
ROW 11 COL 10 'You must also type the employee's number.'

```

```
Employee Menu Help

The Employee System Menu provides the ability to view
or edit an employee's name, social security number,
department, or telephone number.

Type V to view the employee data.

Type E to edit the employee data.

You must also type the employee's number.

F3=Exit
```

Field-specific Help

You can easily implement field-specific help screens using code similar to the previous example combined with the IF CURSOR test:

- The terminal user could press a function key while the cursor is in the field for which he wants help.
- Test each field on your screen to determine the position of the cursor in an AFTER-SCREEN procedure.
- EXECUTE a specific SCREEN activity that displays help for the specific field.

Code the AFTER-SCREEN procedure like this:

```
AFTER-SCREEN. PROC
  IF KEY-PRESSED = F1
    IF EMP# CURSOR
      EXECUTE EMP#-HELP
    ELSE-IF CURSOR NAME
      EXECUTE NAME-HELP
    ELSE-IF CURSOR SSN
      EXECUTE SSN-HELP
    ELSE-IF CURSOR TELEPHONE
      EXECUTE TELEPHONE-HELP
    END-IF
  RESHOW
END-IF
```

Windowed Screens

CA-Easytrieve allows you to code pop-up windows or dialog boxes in your screen declarations. Windows are easily coded by executing one screen activity from another in which the second screen is smaller than the first. The following example illustrates a pop-up window in which the terminal user is asked to confirm an update request before the update is actually performed.

```

...
SCREEN NAME EMPLOYEE-UPDATE LINESIZE 80 ROWCOUNT 24
KEY F3 NAME 'Exit' EXIT
KEY F6 NAME 'Update'
TITLE 1 'EMPLOYEE RECORD'
ROW 3 'EMPLOYEE NUMBER' COL 24 EMP#
ROW 6 'SOCIAL SECURITY NUMBER' SSN
ROW 8 'LAST NAME' COL 24 NAME-LAST
ROW 10 'FIRST NAME' COL 24 NAME-FIRST
...
AFTER-SCREEN. PROC
EXECUTE CONFIRM-WINDOW
END-PROC
SCREEN NAME CONFIRM-WINDOW LINESIZE 40 ROWCOUNT 10 ROW 12 COL 20 +
BORDER (SINGLE)
KEY F6 NAME 'Update'
KEY F12 NAME 'Cancel' EXIT IMMEDIATE
TITLE 1 'Confirm Update'
ROW 3 'Are you sure?'
ROW 5 'Press F6 to update record'
ROW 6 'Press F12 to cancel update'
AFTER-SCREEN. PROC
MOVE PERSNL TO WORK-AREA
READ PERSNL KEY EMP#
MOVE WORK-AREA TO PERSNL
WRITE PERSNL UPDATE
MESSAGE 'Record updated.' LEVEL INFORMATION
EXIT
END-PROC

```

The first screen is defined as 24 rows by 80 columns. When the user presses F6 to update the record, a second SCREEN activity, CONFIRM-WINDOW, is executed. CONFIRM-ACTIVITY is defined as 10 rows by 40 columns and the upper-left corner is located in row 12 column 20. This second screen overlays the first and waits for the user to confirm the request before updating the record. The screen looks like this:

```

                                EMPLOYEE RECORD
EMPLOYEE NUMBER      00370
SOCIAL SECURITY NUMBER 256-52-8737
LAST NAME            NAGLE
FIRST NAME           MARY

                                Confirm Update
                                Are you sure?
                                Press F6 to update record
                                Press F12 to cancel update
                                F6=Update  F12=Cancel

F3=Exit  F6=Update
```

Action Bar Pull-Downs

You can use an action bar with a pull-down menu in your application by invoking overlay screens. The following screen has an action bar at the top of the screen:

```

                                EMPLOYEE RECORD

Options
+-----+
EMPLOYEE NUMBER      00370
SOCIAL SECURITY NUMBER 256-52-8737
LAST NAME            NAGLE
FIRST NAME           MARY
...

F3=Exit  F10=Pulldown options
```

When the user presses F10, the following pull-down menu is displayed:

EMPLOYEE RECORD	
Options	
Find>>	
Print	00370
	BER 256-52-8737
F3=Exit	NAGLE
FIRST NAME MARY	
...	
F3=Exit F10=Pulldown options	

The following is the program code used for this example.

```

FILE PERSNL F(150) INDEXED UPDATE
%PERSNL
ACTION-BAR S 7 A VALUE 'Options'
DASH-LINE S 17 A VALUE '+-----+'
FIND-ACTION W 8 A VALUE 'Find>>' RESET
PRINT-ACTION W 8 A VALUE 'Print' RESET
SCREEN LINESIZE 80 ROWCOUNT 24
KEY F3 NAME 'Exit' EXIT
KEY F10 NAME 'Pulldown options'
TITLE 1 'EMPLOYEE RECORD'
ROW 3 COL 2 ACTION-BAR ATTR (WHITE PROTECT)
ROW 4 COL 2 DASH-LINE ATTR (WHITE PROTECT)
ROW 6 'EMPLOYEE NUMBER' COL 24 EMP#
...
AFTER-SCREEN. PROC
EXECUTE PULLDOWN-WINDOW
END-PROC
SCREEN NAME PULLDOWN-WINDOW LINESIZE 17 ROWCOUNT 7 ROW 4 COL 2 +
BORDER (SINGLE ATTR (WHITE))
KEY ENTER
KEY F3 NAME 'Exit' EXIT IMMEDIATE
ROW 1 FIND-ACTION
ROW 2 PRINT-ACTION
AFTER-SCREEN. PROC
IF FIND-ACTION CURSOR
EXECUTE FIND-WINDOW
ELSE
EXECUTE PRINT-JOB
END-IF
EXIT
END-PROC
SCREEN-NAME FIND-WINDOW LINESIZE 27 ROWCOUNT 10 ROW 4 COL 18 +
BORDER (SINGLE ATTR (WHITE))
KEY ENTER
KEY F3 NAME 'Exit' EXIT IMMEDIATE
TITLE 'Find'
ROW 3
'Key. . . .' EMP#
AFTER-SCREEN. PROC
READ PERSNL KEY EMP# STATUS
...

```


Graph Processing

Overview

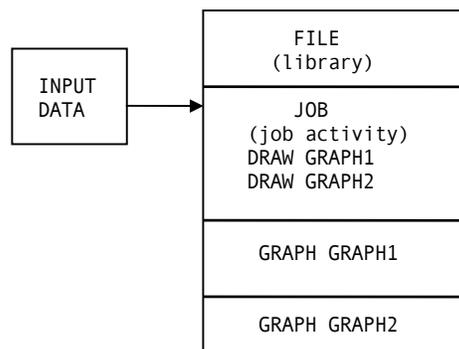
CA-Easytrieve provides a facility for producing bit-mapped presentation graphs with a non-procedural technique similar to reporting. The styles of graphs available include pie charts, bar charts, line graphs, and scatter diagrams. The graph facility controls graph format making assumptions based on best-fit.

Note: Graph processing is available only when using CA-Easytrieve/Workstation.

Basic Structure

The CA-Easytrieve graph facility is fully declarative; you need only define the style and content of the graph, and CA-Easytrieve creates the necessary instructions to produce it. The following exhibit illustrates the basic structure of a CA-Easytrieve JOB with graph processing. You can define one or more graphs for each JOB activity.

CA-Easytrieve Program



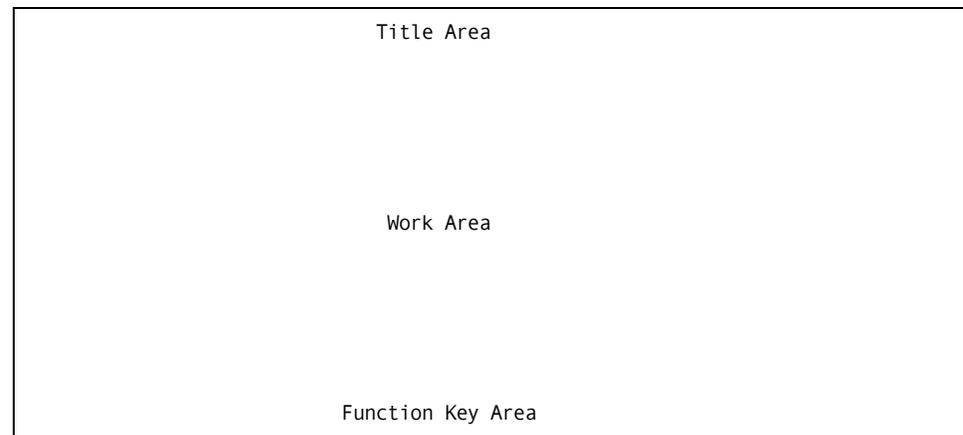
DRAW Statement Processing

The DRAW statement activates the graph logic defined by GRAPH declarations. CA-Easytrieve extracts the data required for the requested graph, formats it in the specified manner, and sends it to the terminal for display and, optionally, printing.

The immediate result of a DRAW statement is to store the data in a work or spool file. The normal termination of each JOB activity includes the processing of any graph work files created during the JOB activity. At this time, each graph is displayed at the terminal in the order in which the GRAPH subactivity is defined. If the JOB activity abends or terminates due to a STOP EXECUTE statement, the graph work files are not processed.

Graph Format

The CA-Easytrieve graph display format is illustrated below:



Title Area

The optional title area consists of a single line designated as the title by a TITLE statement in the graph declaration.

Work Area

The work area contains the display of data points specified on the VALUE statement. The y-value data points are optionally summarized and categorized by x-value. The work area also displays a legend identifying the data. CA-Easytrieve fields are identified by their headings. The following rules control the heading; the order in which they are listed indicates the hierarchy of override:

1. The NOHEADING parameter of the GRAPH statement inhibits the printing of any headings.
2. The HEADING statement sets the item heading content.
3. The HEADING parameter of the DEFINE statement sets the item heading content.
4. The *field-name* of the DEFINE statement sets the item heading content.
5. Numeric literals used as graph values do not have headings. Numeric literals are useful for displaying a graph having a count of categorized records.

Function Key Area

The function key area shows the system-defined function key assignments for the graph view facility. Using these keys you can receive help, exit the view, or print the graph to the default print device.

GRAPH Statement

You define a graph in CA-Easytrieve by coding a GRAPH statement followed by a series of graph definition statements. You must code the GRAPH statement first in a GRAPH declarative. The GRAPH statement establishes the style and characteristics of the graph. See Graph Statement in the *CA-Easytrieve Language Reference Guide* for complete explanations of GRAPH statement parameters.

Graph Definition Statements

A set of graph definition statements defines every CA-Easytrieve graph. The statements define the graph style, format, sequence, and data content. Graph definition statements are the last statements coded in a JOB activity. The statements are combined to make a subactivity. All subactivities (REPORTs and GRAPHs) must appear at the end of the JOB activity. The graph definition statements must be coded in the order as shown below.

```

...
JOB ...
...
DRAW ...
...
GRAPH
Graph  {SEQUENCE
Defini {TITLE
tion   {HEADING
Statem {VALUE
ents

```

- SEQUENCE - optionally specifies the order of the graph values. You would normally sequence the values by the x-value.

- TITLE - defines optional title items and their position on the title line.
- HEADING - optionally defines an alternative heading or label for a field.
- VALUE - defines the content of the graph. The x-value is used as the horizontal axis of the graph. One or more numeric y-values are used on the vertical axis of the graph.

See the *CA-Easytrieve Language Reference Guide* for complete syntax.

Processing a Graph

CA-Easytrieve performs the following steps to produce graphs:

Step 1: For each DRAW statement executed, a spool file record is written to a temporary work file. The spool file record contains the contents of all the fields coded in the GRAPH subactivity.

Step 2: At normal termination of the JOB activity, each graph spool file is closed, optionally sequenced, and re-opened.

Step 3: A temporary graph metafile is created. This file becomes the input to the Graph Display Facility. The Graph Display Facility actually displays the graph.

Step 4: Each record from the spool is read and the sequence and title fields are discarded. The x-value and y-value fields are written to the metafile.

Step 5: The graph spool file is closed and deleted.

Step 6: The metafile is closed.

Step 7: The Graph Display Facility is invoked to display the graph via an MS-DOS shell. This facility reads the metafile and performs the summation of the y-values for duplicate x-values if SUMMARY is specified on the GRAPH statement.

Step 8: The graph is displayed on the terminal.

Step 9: If you press F6, the graph is printed to the default printer device.

Note: The MS-DOS GRAPHICS.COM program must be loaded before printing a graph.

Step 10: When you press F3, the Graph Display Facility terminates and deletes the metafile.

Step 11: Program execution continues.

Sample Graph Applications

Following are coding examples for several common graph applications. Included application examples are:

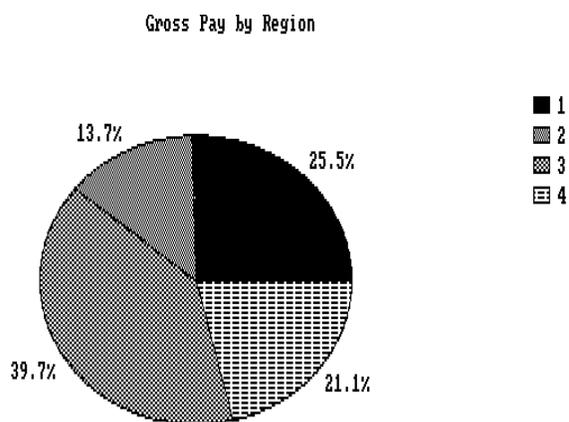
- Pie chart
- Bar charts
- Line chart
- Scatter diagram.

Pie Chart

The following program produces a pie chart showing the summary of gross pay categorized by region. If you do not specify a STYLE on the GRAPH statement, CA-Easytrieve defaults to a pie chart.

```
FILE PERSNL
%PERSNL
JOB INPUT PERSNL NAME DISPLAY-GRAPH
DRAW PAY-BY-REGION
GRAPH PAY-BY-REGION SUMMARY
SEQUENCE REGION
TITLE 'Gross Pay By Region'
VALUE REGION PAY-GROSS
```

The chart created appears below:



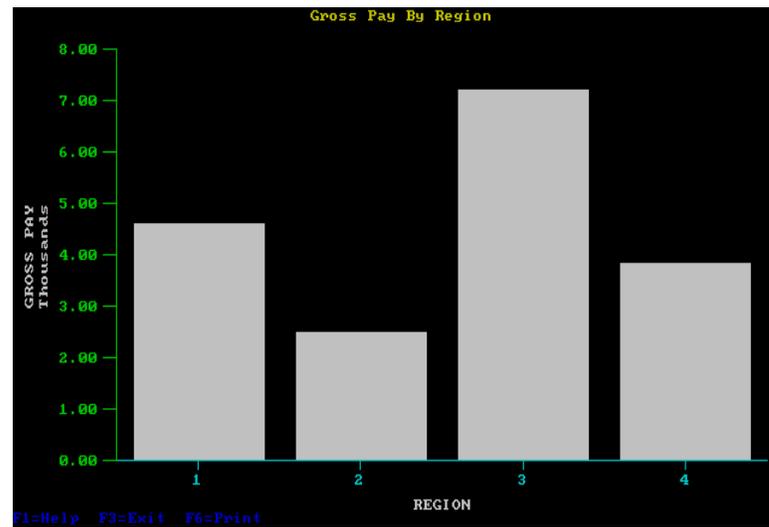
F1=Help F2=Keys F3=Exit F4=Shell F6=Print F12=Cancel

Vertical Bar Chart

The following program produces a vertical bar chart showing the summary of gross pay categorized by region.

```
FILE PERSNL
%PERSNL
JOB INPUT PERSNL NAME DISPLAY-GRAPH
DRAW PAY-BY-REGION
GRAPH PAY-BY-REGION SUMMARY STYLE 'VBAR'
  TITLE 'Gross Pay By Region'
  VALUE REGION PAY-GROSS
```

The chart created appears below:

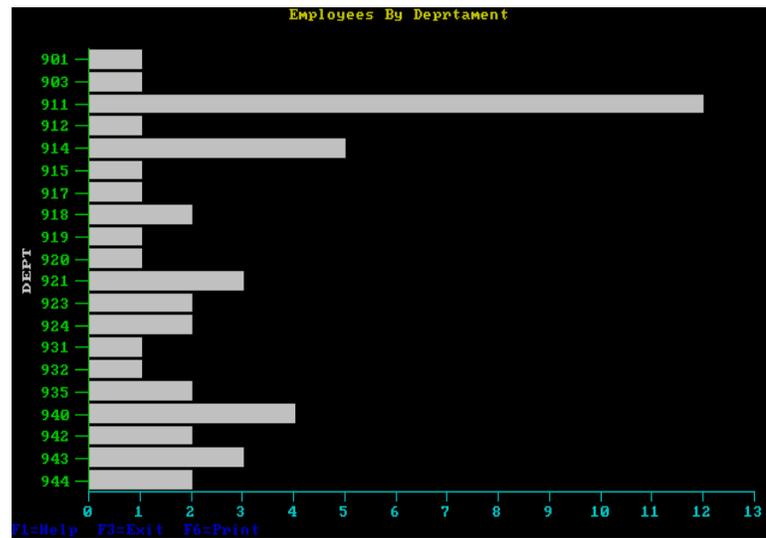


Horizontal Bar Chart

The following program produces a horizontal bar chart showing the count of employees by department. The numeric literal 1 is used to count the employees.

```
FILE PERSNL
%PERSNL
JOB INPUT PERSNL NAME DISPLAY-GRAPH
DRAW EMPLOYEES-BY-DEPARTMENT
GRAPH EMPLOYEES-BY-DEPARTMENT SUMMARY STYLE 'HBAR'
TITLE 'Employees By Department'
VALUE DEPT 1
```

The chart created appears below:

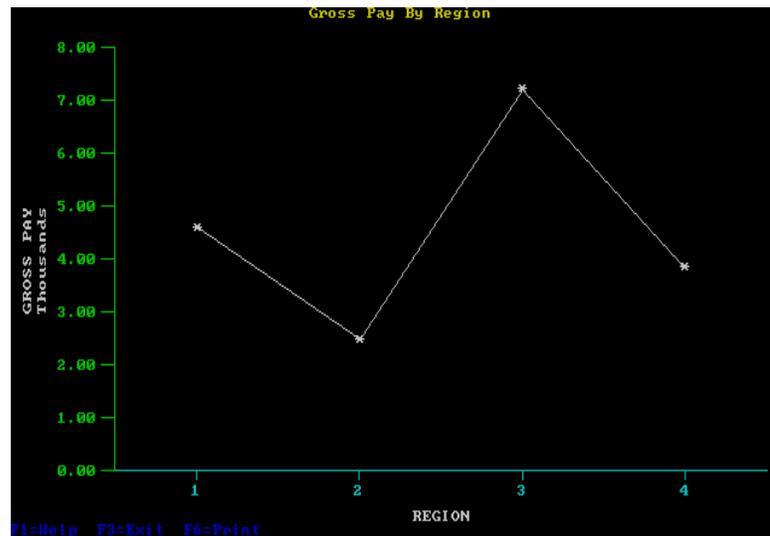


Line Chart

The following program produces a line chart showing the summary of gross pay categorized by region.

```
FILE PERSNL
%PERSNL
JOB INPUT PERSNL NAME DISPLAY-GRAPH
DRAW PAY-BY-REGION
GRAPH PAY-BY-REGION SUMMARY STYLE 'LINE'
TITLE 'Gross Pay By Region'
VALUE REGION PAY-GROSS
```

The chart created appears below:

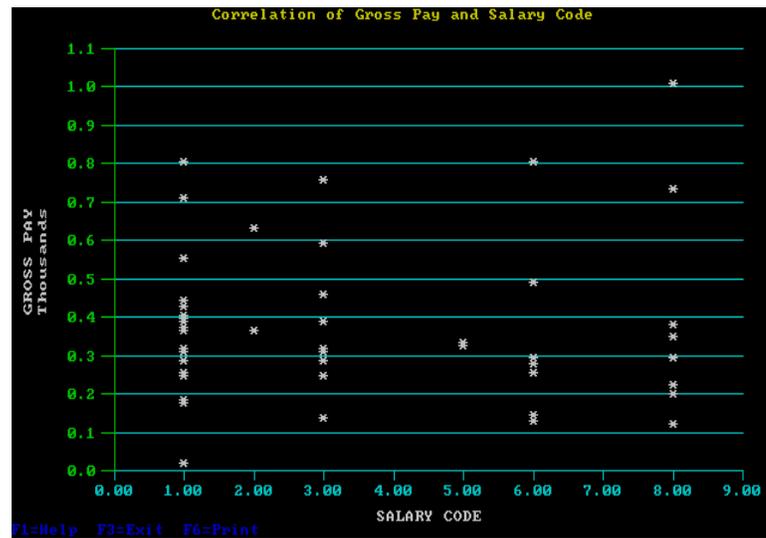


Scatter Diagram

The following program produces a scatter diagram showing the correlation between gross pay and salary code.

```
FILE PERSNL
%PERSNL
JOB INPUT PERSNL NAME DISPLAY-GRAPH
DRAW GROSS-VS-SALARY-CODE
GRAPH GROSS-VS-SALARY-CODE STYLE 'SCATTER'
TITLE 'Correlation of Gross Pay and Salary Code'
VALUE SALARY-CODE PAY-GROSS
```

The diagram created appears below:



CA-Easytrieve Macro Facility

The Macro Facility permits you to easily duplicate often-repeated source statements in any program. This enables you to tailor CA-Easytrieve to the programming standards of your installation.

Even the most inexperienced programmer can effectively use CA-Easytrieve macros. The macro library allows you to store the data definition statements of frequently used files using standardized data-naming conventions.

This chapter first discusses how macros are invoked using the macro invocation statement. Second, it discusses the two parts of a macro:

- The macro prototype statement
- The macro body.

This chapter concludes with a description of macro processing and parameter substitution.

Macro Invocation Statement

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

Syntax

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

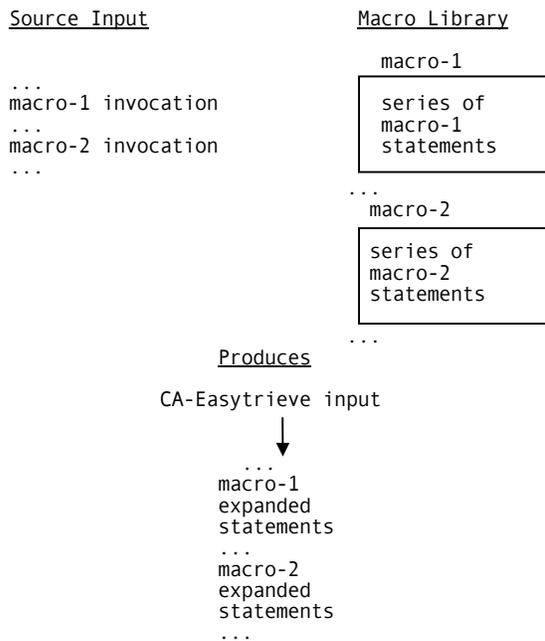
macro-name Supply the name of a previously stored macro that you want to invoke.

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

keyword-parameter Supply both the keyword and values of keyword parameters in the macro.

Invoking Macros

To invoke a macro, you code a macro invocation statement anyplace within the CA-Easytrieve source program. Macro invocation statements cause a series of statements to be retrieved from the macro library and included as source statements in the CA-Easytrieve program. The series of statements can be modified by parameters supplied on the macro invocation statement.



Macro Library

Mainframe macro statements are stored and maintained in a macro library. Each macro library is connected by your system administrator. You can enable or disable the libraries as required. See the *CA-Easytrieve/ESP User Guide* for more information.

Macro Files

UNIX and Workstation macro statements are stored as files. Each macro is stored in a file named *xxxxxxx.mac* where *xxxxxxx* is the macro's name. See the *CA-Easytrieve/Workstation User Guide* or the *CA-Easytrieve for UNIX User Guide* for more information.

Macro Library Security

CA-Easytrieve allows you to protect your macro statements with the use of an ACCESS record. 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. Refer to the *CA-Easytrieve/Online Administrator Guide* for detailed information about creating and maintaining macro libraries.

CA-Panvalet

In addition to having the maintenance and backup capabilities provided by CA-Panvalet, CA-Easytrieve gives you the ability to secure the macro against unauthorized access. This is accomplished through a security access code which can be applied to a CA-Panvalet member.

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. The syntax is:

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

VSAM

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. The syntax is:

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

Macro Definition

Macros consist of three parts:

1. The macro prototype, which defines the parameters of the macro.
2. The macro body, which contains the CA-Easytrieve statements to be generated by a macro invocation statement.
3. The optional macro termination command.

A macro's name is the same as the member name in the macro storage library. The following illustrates the parts of a macro:

```

PROTOTYPE STATEMENT [ MACRO 2 NUMBER RESULT
                        [
BODY                [ *****
                        [ *
                        [ * NAME:  MACRO EXAMPLE          *
                        [ *          CALCULATE THE CUBE OF A NUMBER      *
                        [ *
                        [ * FUNCTION: THIS MACRO CALCULATES THE CUBE OF A NUMBER.*
                        [ *
                        [ *****
                        [ DEFINE CUBE_NUMBER_  S      6  N VALUE 000000
                        [ CUBE_NUMBER_ = &NUMBER * &NUMBER * &NUMBER
                        [ &RESULT = CUBE_NUMBER_
Optional Termination Command [ MEND
    
```

Macro Prototype Statement

The prototype statement must be the first statement of a macro. It optionally defines the parameters of the macro. Either positional and/or keyword parameters can be used.

Positional Parameters

Use positional parameters when a value is always required for the parameter each time the macro is invoked. Frequently used parameters are often positional, since you need to code only the value of the parameter.

Keyword Parameters

Use keyword parameters:

- To help keep track of a large number of parameters
- To specify optionally-used parameters
- To specify a default value for parameters.

Prototype Examples

Following are examples of macro prototype statements. The first example shows a macro with no substitution parameters:

```

MACRO
...
...
    
```

This example shows a macro with only positional parameters. The number of positional parameters is not indicated. You could have coded the optional parameter as a 2.

```
MACRO POS1 POS2  
...  
...
```

This example shows a macro with only keyword parameters. Here you code the number of positional parameters as zero. This is a required parameter when you use keyword parameters.

```
MACRO 0 KEY1 VALUE1 KEY2 VALUE2  
...  
...
```

This example shows a macro with positional and keyword parameters. Here you code the number of positional parameters as a 1.

```
MACRO 1 POS1 KEY1 VALUE1  
...  
...
```

Macro Body

The macro body consists of a series of model and actual CA-Easytrieve statements. The model statements contain one or more parameters that are replaced by the values of corresponding parameters on the prototype statement.

Macro Termination Command

The optional macro termination command is used at the end of a macro. Its syntax is:

```
MEND
```

Macro Processing

Macro processing occurs whenever a macro invocation statement appears in a CA-Easytrieve program. You designate a macro invocation by prefixing a '%' (percent sign) to the macro name. Each macro invocation retrieves a copy of the macro from the library and, if necessary, replaces parameters with their corresponding values from the macro invocation statement or the prototype statement.

Parameter Substitution

The rules for substituting macro parameters are the basic rules-of-syntax and the following:

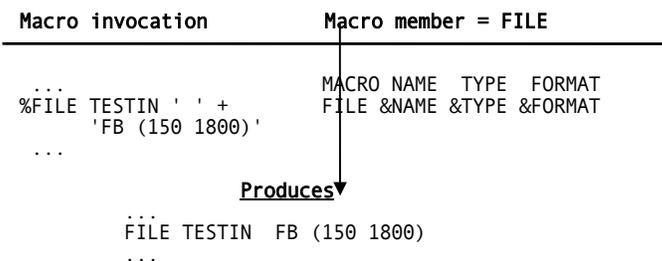
- You must specify positional parameter values on the macro invocation statement in the same order that they appear on the prototype statement.

- CA-Easytrieve gives the value of a null string to unsupplied positional parameter values. The parameter is treated as nonexistent.
- You can specify keyword parameter values in any order on the macro invocation statement.
- CA-Easytrieve gives unsupplied keyword parameter values the default value from the prototype statement.
- Within the body of a macro, the ampersand (&) is the prefix concatenated to parameter substitution words. You must spell parameter substitution words exactly like their counterparts on the macro prototype except for the leading ampersand. Delimit parameter substitution words with a space or a period (.). Use a period when the substituted value is to be concatenated with another word. CA-Easytrieve deletes the period when the parameter is replaced by its value. When you want to use an ampersand in the body of the macro, you must code two consecutive ampersands (&&).

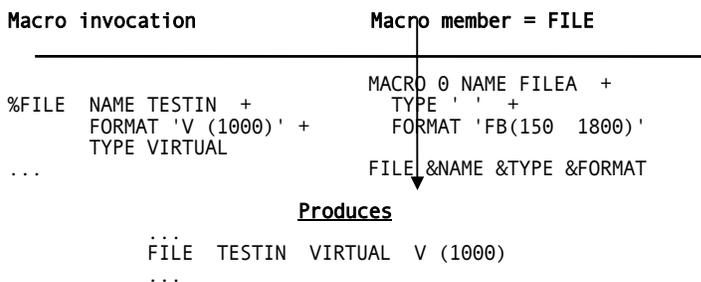
CA-Easytrieve treats a macro invocation statement within the body of a macro (nested) as if it were outside of the macro. Any number of nesting levels can be used.

Examples

The following example illustrates positional parameter substitution. The second parameter value (' ') is supplied only to maintain correct positioning for the third parameter ('FB (150 1800)').



The next example illustrates keyword parameter substitution. The default value of a space for the second keyword entry (TYPE) is an efficient way to code parameters that are used infrequently.



Parameter Substitution in a Macro

This example illustrates the use of the ampersand within a macro body statement and concatenated substitution words. The extra ampersand and the periods used for concatenation are not part of the resulting statements.

<pre>Macro invocation ... %FILE TESTIN NEW ...</pre>	<pre>Macro member = FILE ----- MACRO NAME PREFIX FILE &NAME &PREFIX.-SSN 1 9 N &PREFIX.-MAIL 10 75 A, + HEADING 'NAME && ADDRESS'</pre>
	↓
	Produces
	<pre>... FILE TESTIN NEW-SSN 1 9 N NEW-MAIL 10 75 A, + HEADING 'NAME & ADDRESS' ...</pre>

Instream Macros

Macro statements can also be included in the source input to CA-Easytrieve. This capability is particularly useful for testing new macros prior to storing them in the macro library. When an instream macro has the same name as a macro in the library, the instream macro is used.

Instream macros are placed at the beginning of the source input prior to *any* other statements. Each instream macro is bounded by an MSTART and MEND statement. The format of these statements is:

```
MSTART macro-name
MACRO 2 NUMBER RESULT
*****
*                                     *
* NAME:  MACRO EXAMPLE                 *
*       CALCULATE THE CUBE OF A NUMBER *
*                                     *
* FUNCTION: THIS MACRO CALCULATES THE CUBE OF A NUMBER. *
*                                     *
*****
DEFINE CUBE NUMBER_ S 6 N VALUE 00000
CUBE_NUMBER_ = &NUMBER * &NUMBER * &NUMBER
&RESULT = CUBE_NUMBER_
MEND
```

Macro-name is the name of the macro. It can be from one to eight characters long. The first character must be alphabetic.

Example

This example illustrates the use of instream macros.

Statements:

```
MSTART EXMACRO
MACRO 2 NUMBER RESULT
PUSH
SKIP 1
SKIP 1
LIST OFF
*****
*
* NAME:  MACRO EXAMPLE
*       CALCULATE THE CUBE OF A NUMBER
*
* FUNCTION:  THIS MACRO CALCULATES THE CUBE OF A NUMBER.
*
*****
POP
SKIP 1
DEFINE CUBE_NUMBER_  S  6  N VALUE 000000
SKIP 1
  CUBE_NUMBER_ = &NUMBER * &NUMBER * &NUMBER
  &RESULT = CUBE_NUMBER_
SKIP 1
MEND
*
DEFINE CUBED_RESULT  W  6 N VALUE 000000 MASK (J 'ZZZZZ9')
JOB INPUT NULL  NAME MACRO1
%EXMACRO 3 CUBED_RESULT
  DISPLAY CUBED_RESULT
STOP
```

Produce:

27

Glossary

activity section

A required section of a CA-Easytrieve program in which the executable statements (procedural or declarative) that process your data are coded.

alphanumeric literal

A word enclosed in single quotes, up to 254 characters long, consisting of alphabetic characters A through Z, and numeric characters 0 through 9.

arithmetic expression

An expression that enables two or more numeric quantities to be combined to produce a single value.

array

A series of consecutive memory locations in one or more dimensions.

assignment

Establishing the value of a field as a result of simple data movements, an arithmetic expression, or logical bit manipulation.

bounds checking

A process that automatically checks that indexes and subscripts do not reference data outside the storage boundary of the field being referenced.

commit point

A point at which updates are committed to the operating system.

commit processing

A process that issues commands to the operating environment signifying the end of one unit of work and the start of the next.

conditional expression

Program logic using a statement that offers an alternative to the normal top-to-bottom execution of a program.

control report

A report that automatically accumulates and prints totals of quantitative report information.

conversational program

In a CICS environment, a program in which CA-Easytrieve does not issue a commit point during each terminal I/O. See SCREEN COMMIT Parameter in the "Screen Processing" chapter for more information.

decision/branching logic

Controlling the execution of a program by coding statements that govern the flow of a program depending on the truth value of a conditional expression.

double-byte character set (DBCS)

A character representation for a national language, such as Japanese, where the number of characters used by the national language cannot be represented by a single byte character set (SBCS), such as EBCDIC or ASCII.

environment section

An optional section of a CA-Easytrieve program that enables you to customize the operating environment for the duration of a program's compilation and execution by overriding selected general standards. See the PARM Statement.

external table

Table data that is located in a file external to the program. An external table is established just before use.

graph definition statements

A set of CA-Easytrieve statements that defines a graph style, format, sequence, and data content.

graph format

The display format of a CA-Easytrieve graph, consisting of a title area, work area, and function key area.

GRAPH subactivity

An area in a JOB activity where graphs are described.

hexadecimal literal

A word used to code a value that contains characters not available on standard data entry keyboards. A hexadecimal literal must be prefixed with X', and terminated with a single quote, and can contain digits 0 through 9, and letters A through F.

indexing

A data reference that results from CA-Easytrieve deriving a displacement value to correspond to a particular occurrence in a field name defined with OCCURS. See Array Processing for more information.

input edit pattern

A sequence of characters that describes the format of the data in the field.

instream macro

Macro statements that are included in the source program input to CA-Easytrieve.

instream table

Table data within the program immediately following the argument (ARG) and description (DESC) fields for the file. An instream table is established at the time the program is compiled.

JOB activity

An activity in a CA-Easytrieve program that reads information from files, examines and manipulates data, writes information to files, and initiates reports and graphs.

label report format

An alternate CA-Easytrieve report format that is used to print labels.

library section

An optional section of a CA-Easytrieve program that describes the data to be processed by the program. A library section is optional only if a program is not doing any input or output of files.

macro

One or more often-repeated CA-Easytrieve source statements that you can save to use in more than one program or multiple times within a program.

mask

A user-supplied sequence of characters against which CA-Easytrieve edits data input into a numeric field.

MIXED format literal

A word that contains both DBCS and SBCS characters.

native SQL statements

CA-Easytrieve SQL statements, prefixed by the SQL keyword, that are equivalent to many of those available in COBOL.

null data value

A value that denotes the absence of a known value for a field.

pattern

A user-supplied sequence of characters that describes the format of data in a field. Used to edit complex combinations of data types and character sequences.

PROGRAM activity

An activity in a CA-Easytrieve program that is a simple top-down sequence of instructions.

pseudo-conversational program

In a CICS environment, a program in which CA-Easytrieve issues a commit point during each terminal I/O. See SCREEN COMMIT Parameter in the “Screen Processing” chapter for more information.

report definition statements

A set of CA-Easytrieve statements that define the report type, format, sequence, and data content of a report.

report procedures

Routines provided in CA-Easytrieve that perform special data manipulation for a report.

REPORT subactivity

An area in a JOB activity where reports are described.

rollback

A process that recovers the updates made since the last commit point.

S (static) working storage field

A field that is stored in a static working storage area and is not copied onto report work files.

screen attributes

Information that controls the display of screen items, such as color and brightness.

screen definition statements

A set of CA-Easytrieve statements that define screen format and data content.

screen format

The layout of a CA-Easytrieve screen that consists of a title area, work area, message area, function key area, and border. The size of the screen defaults to the values specified in the Site Options Table.

screen item

A field or literal that you want to display to or receive from the terminal user.

screen procedures

Routines provided in CA-Easytrieve that perform customized actions specific to your screen application.

SCREEN activity

An activity in a CA-Easytrieve program that defines a screen-oriented transaction.

segmented data

A common data structure in which each record contains a fixed portion of data and multiple occurrences of data segments.

sequenced report

A report in which data items used in the report are sorted.

signed (quantitative) field

A numeric field with decimal positions (0 to 18).

single-byte character set (SBCS)

A character representation where each character occupies a single byte of storage.

Site Options Table

A table installed with CA-Easytrieve containing options that control the compilation and execution of your programs. You can customize the Site Options Table to your specific requirements.

SORT activity

An activity in a CA-Easytrieve program that creates sequenced or ordered files.

standard report format

The CA-Easytrieve default format of a report that consists of a top margin, an optional title area, an optional heading area, the report body, and a bottom margin.

state tables

Examine each character in a string to determine if the string is acceptable or not. Each character examined causes a transition from one state to the next. Most compilers use state tables for recognizing tokens, such as labels or identifiers.

subscript

An integer (or a field containing an integer) that represents the occurrence number of the element within the array to be referenced.

Synchronized File Processing (SFP)

A facility in CA-Easytrieve that performs match/merge operations on multiple files, or compares the contents of a key field or fields from one record to the next in a single file.

SYSPRINT

The default CA-Easytrieve system output printer, whose actual destination is set in the Site Options Table.

system default masks

Edit masks supplied by CA-Easytrieve.

system-defined fields

Special read-only data fields provided by CA-Easytrieve that are stored as part of working storage.

table

A collection of uniform data records. The *argument* uniquely identifies a table entry; the *description* is the remainder of the table entry.

unsigned (non-quantitative) field

A numeric field with no decimal positions.

Virtual File Manager (VFM)

A CA-Easytrieve sequential access method that processes work files needed by a program.

W (work) working storage field

A field that is copied onto the report work files at the time a PRINT statement is executed.

Index

%

% (percent sign) 10-5

%macro-name 10-1

&

& (ampersand) 10-6, 10-7

/

/C (compiler) compiler switch 2-73, 2-76

/FR (Far Reference) compiler switch 2-73, 2-76, 2-77

A

ACCESS record 10-3

ACTION message level 8-26, 8-28

ACTIVITY parameter 2-26

Activity section 2-3

AFTER-BREAK report procedure 7-40

AFTER-LINE report procedure 7-38

AFTER-SCREEN screen procedure 8-31, 8-33, 8-34, 8-35

Alphanumeric editing 8-17

Alphanumeric literals 2-14

Arithmetic expressions 2-32

evaluating 2-33

Arithmetic operators 2-32

Array processing 2-44

bounds checking 2-45

multiple dimension arrays 2-46

single dimension arrays 2-45

subscripting 2-49, 2-50

subscripts 2-78

Arrays

displaying on a screen 8-27

two-dimensional 8-28

ASCII

alpha (A) data format 2-17

alphanumeric literals 2-14

data formats 2-15

hexadecimal literals 2-14

Assembler calling convention 2-60

Assembler subprogram linkage 2-57

Assignment statement 2-7, 2-31, 2-35

mainframe EBCDIC to DBCS conversion 2-36

rules 2-36

Assignments and moves 2-31

ATTR parameter 2-12, 8-5, 8-8, 8-26

Automatic editing

against values 8-25

alphanumeric 8-16

numeric 8-15

Automatic file input 3-3

IDMS interface 5-1

IMS/DLI interface 6-1, 6-4

Axis, graph 9-4

B

Bar chart example 9-6, 9-7

BASIC calling convention 2-60
BEFORE procedure 3-22
BEFORE-BREAK report procedure 7-39
BEFORE-LINE report procedure 7-38
BEFORE-SCREEN screen procedure 8-31, 8-33
Binary fields 2-78
Binary numbers 2-7, 2-8
Borders, screen 8-3
Bounds checking 2-45
 on the workstation 2-79
Branch actions 8-33
Branching logic 2-29
BREAK-LEVEL field 7-8
BTRIEVE 3-33
BWZ (blank when zero) 8-16

C

C calling convention 2-60
CA-Easytrieve publications 1-3
CA-Easytrieve/ESP interactive execution 2-77
CA-IDMS *See* IDMS interface
CALL statement 2-55
 stack usage on workstation 2-61
CA-Panvalet member security 10-3
CARD files 3-4
CARD input of sequential files 3-11
CA-SuperCalc 3-10, 3-35
Child programs 2-71
CICS
 coding efficient programs 2-79
 printer files 3-28
 pseudo-conversational programs 8-36
 sort program 3-21
 SYNCPOINT command 2-28
 SYNCPOINT ROLLBACK command 2-28
 terminal identification 8-7
C-ISAM files 2-29, 3-38

CLOSE statement 4-18
COBOL subprogram linkage 2-58
Code system 2-15, 2-16
 DBCS 2-18, 2-19, 2-20
Comma-delimited files 3-10, 3-35
Commit processing 2-25, 8-40
 automatic 2-26
 commit points 2-25
COMMIT statement 4-4
Communications block (IDMS interface) 5-11
Compiler options, efficient use 2-78
Compiler switches 2-76
 /C (compiler) 2-73
 /FR (Far Reference) 2-73
Conditional expressions 2-30
 combining 2-31
 DBCS and MIXED field support 2-31
 EBCDIC to DBCS conversion 2-31
 synchronized file processing 3-26
CONNECT statement 4-18, 4-24
Control break 7-18
CONTROL reports
 summary file 7-32
CONTROL Reports
 annotating 7-40
 referencing data 7-18
CONTROL statement 7-7, 7-17
 label reports 7-16
Controlled file input
 IMS/DLI interface 6-9
Controlled input processing 3-3
Controlled processing, IDMS interface 5-1
Conventions, syntax 1-5
Conversion rules
 mainframe 2-18
 workstation 2-14
COPY statement 5-15
Create mode 3-7
CURSOR attribute 8-26
CURSOR statement 8-26
Customization, environment 2-3

D

- Data availability tests 3-6
- Data buffering mode 3-3
- Data description 2-3
- Data formatting
 - ASCII 2-15
 - EBCDIC 2-15
 - rules 2-13
- Data references 2-8
- Data strings, evaluating 2-53
- Data, segmented 2-51
- Database records
 - use of 5-15
- DBASE 3-10, 3-33
- DBCS
 - code system 2-18, 2-19
 - data format 2-20
 - format literals 2-22
- Decision and branching logic 2-29
- DECLARE statement 2-12, 2-13, 8-5
- DEFAULT FIELD ERROR statement 8-26
- DEFAULT FIELD statement 8-10
- DEFAULT KEY statement 8-30
- DEFAULT LITERAL statement 8-10
- DEFAULT MESSAGE statement 8-29
- DEFAULT TITLE statement 8-8
- DEFINE statement 2-5, 4-17, 4-23, 8-13, 9-3
 - HEADING parameter 7-11
- Defining fields 2-5
- Defining files 2-5
- Development process 2-1
- Device types 3-28
- Direct access mode 3-8
- DISPLAY HEX 2-11
- DISPLAY statement 7-1, 7-35
- Distributable applications 1-11
- DO statement 2-30

- Documentation conventions 1-5
- DRAW statement 9-2, 9-4
- DTLCOPY subparameter 7-27
- DTLCOPYALL subparameter 7-28
- DTLCTL parameter 7-23
- Dynamic access mode 3-8, 3-9

E

- EBCDIC
 - alpha (A) data format 2-17
 - alphanumeric literals 2-14
 - data formats 2-15
 - hexadecimal literals 2-14
- Edit masks 8-13, 8-15, 8-16
 - hexadecimal 8-14
 - patterns 8-16
- Edit patterns 2-13
- Element record definitions, IDD interface 5-8
- Element records, use of 5-10, 5-15
- ENDPAGE report procedure 7-41
- END-REPEAT statement 8-2, 8-10, 8-27
- ENDTABLE statement 2-43
- Environment section 2-3
- EOF processing 4-13
- Error condition handling
 - mainframe 2-59
 - workstation 2-65, 2-70
- Error conditions 3-6
 - screens 8-25
- Error messages, subprogram exits 2-65
- EXIT action 8-30, 8-33, 8-36
- Exit parameter list 2-62
- Exits 2-62
 - error condition handling 2-59
 - file 2-54, 2-55
 - FILE EXIT Linkage 2-66
 - file exits example 2-63
- Explicit connect 4-7

Expressions, arithmetic 2-32

Extended reporting 7-46

F

FABS ISAM files 3-14

on the workstation 3-32

FAR PASCAL calling convention 2-60

FETCH statement 4-20

Field definitions 2-5

IDD interface 5-8

Fields

assigning and moving 2-11

binary 2-78

comparing 2-12

copying 2-32

defining 2-5

file fields 2-6

integer 2-78

packed decimal 2-78

quantitative 2-7

unsigned (non-quantitative) 2-8

varying length 2-10, 2-11, 2-40

working storage 2-6

File access modes 3-8

File browse mode 3-9

File buffers 3-3

File definitions 2-5, 3-2

IDD interface 5-8

FILE EXIT linkage

MODIFY file exit 2-69

FILE EXIT Linkage 2-66

File exits 2-54, 2-55

example 2-63

File fields 2-6

File input, synchronized 3-22

File processing

modes 3-7

rules 3-3

FILE statement 2-5, 3-2, 3-13, 3-15, 3-28, 7-32

CODE parameter 3-31

EXIT parameter 2-54, 2-59, 2-62

MODIFY subparameter 2-63

on the workstation 3-29

PRINTER device type 3-28

SYSTEM parameter 3-2, 3-29

WORKAREA parameter 3-4, 3-6

Files

BTRIEVE 3-33

CA-SuperCalc 3-35

C-ISAM 3-38

closing 3-7

comma-delimited 3-35

DBASE 3-33

host mainframe 3-36

LOTUS 3-34, 3-36

opening 3-7

UNIX 3-37

FILE-STATUS field 3-5, 3-6

FILL parameter 8-12

Fonts

defined 7-46

font codes 7-49

identification 7-49

Footers

page 7-41

report 7-42

Format relationship rules

mainframe 2-20

workstation 2-15

Format rules

mainframe 2-18

workstation 2-14

FORTRAN calling convention 2-60

Function keys 8-30

branch actions 8-30

IMMEDIATE 8-31

symbolic names 8-31

G

GIVING parameter 2-72, 2-74

GOTO JOB statement 2-2

GOTO SCREEN statement 2-2, 8-33, 8-34

Graph definition statements 9-3

Graph Display Facility 9-4

Graph format

- function key area 9-3
- title area 9-2
- work area 9-2
- Graph processing
 - display format 9-2
 - graph definition statements 9-3
 - metafile 9-4
 - printing 9-4
 - sample applications 9-5
 - steps 9-4
- GRAPH statement 9-3, 9-4
 - NOHEADING parameter 9-3
- GRAPH subactivity 2-4, 2-23, 9-1, 9-4
- GRAPHICS.COM program 9-4

H

- HEADING statement 7-11, 9-3, 9-4
- Hexadecimal literals 2-14
- HLLAPI 2-74, 3-10, 3-36
- Hold/release processing 3-9
- Horizontal axis on graph 9-4
- Horizontal bar chart example 9-7
- Host interface, workstation 2-74
- Host mainframe files 3-36
- Host variables 4-7, 4-12

I

- IDD interface 5-8
 - array definitions 5-9
 - default INDEX name 5-9
 - element record definitions 5-2, 5-8
 - field definitions 5-2, 5-8
 - file definitions 5-2, 5-8
 - logical record definitions 5-2, 5-8
 - program name 5-9
 - qualification 5-9
 - record definitions 5-2, 5-8
 - segmented data 5-9
- IDMS databases *See* IDMS interface

- IDMS interface 5-10
 - automatic input 5-15
 - COMMIT command 2-28, 2-29
 - communications block 5-11
 - control statements 5-22
 - FINISH command 2-28, 2-29
 - IDD control statements 5-8
 - processing on the workstation 5-3
 - ROLLBACK command 2-28, 2-29
 - sample database 5-5
 - sample logical record 5-7
 - Sequential Processing Facility (SPF) 5-1
 - tickler file 5-1, 5-16
- IF CURSOR statement 8-38, 8-50
- IF MODIFIED statement 8-38
- IF statement 2-30
 - synchronized file processing 3-25
- IMMEDIATE keys 8-31, 8-34, 8-35, 8-36, 8-39
- Implicit connect 4-7
- IMS/DLI interface 6-1
 - automatic file input 6-1, 6-4
 - automatic input 6-4
 - complete path processing 6-10
 - control statements 6-1, 6-4
 - controlled processing 6-9
 - database maintenance 6-11
 - input definition 6-5
 - PCB and PSB processing 6-3
 - sweep of database 6-4
 - tickler file control 6-4
 - typical path examples 6-5
- INDEX attribute 2-45
- Index manipulation 2-44
- Indexed files
 - adding records to 3-15
 - creating 3-16
 - deleting a record 3-16
 - random input 3-15
 - updating a record 3-17
- INDEXED files 3-14
- Indexes 2-78
- Indexing 2-10
- INFORMATION message level 8-28
- INITIATION screen procedure 8-31, 8-32
- Input edit patterns 2-13

Input mode 3-7
Input/output exits 2-62
Instream macros 10-7
Instream tables 2-42, 2-43
Integer fields 2-78
Integer numbers 2-8
Interprogram linkage 2-54
Invoking programs 2-71
 other CA-Easytrieve programs 2-54
 subprograms in other languages 2-55

J

JOB activity 2-4
 graph subactivity 9-2
 print file processing 7-4
Job flow 2-24
JOB INPUT SQL statement 4-13, 4-23
JOB INPUT statement 3-27
JOB statement 3-23, 3-24, 4-18
 synchronized file processing 3-24
JUSTIFY parameter 8-13

K

KEY statement 8-3, 8-6, 8-30
KEY-PRESSED field 8-6, 8-31

L

Label reports 7-15
 CONTROL statement 7-16
 truncating 7-16
LEVEL field 7-8, 7-19, 7-40
Library section 2-3
 definition 4-7
Line chart example 9-8
LINE statement 7-12

LINE-COUNT field 7-8
LINE-NUMBER field 7-8
LINK statement 2-55
 HOST parameter 2-74
 implementation 2-73
 vs. CALL statement 2-71
 workstation 2-73
Linkage conventions
 mainframe 2-57
 workstation 2-60, 2-66
Linking subprograms 2-13

Literals

 alphanumeric 2-14
 DBCS format 2-22
 hexadecimal 2-14
 MIXED format 2-22
 types of 2-13

Local Area Networks (LANs) 3-10
Logical record definitions, IDD interface 5-8
Logical records 5-11
 use of 5-10, 5-15
Logical unit of work 2-26
LOTUS 3-10, 3-34, 3-36

M

Macro facility 10-1
Macros 10-1
 body of 10-5
 defining 10-3
 instream 10-7
 invocation statement 10-1
 invoking 10-2
 keyword parameters 10-2
 library 10-2
 nesting 10-6
 parameter substitution 10-5, 10-6
 positional parameters 10-1
 processing 10-5
 protecting 10-3
 prototype statement 10-4
 termination command 10-5
 three parts of 10-3
Mailing labels 7-15

Mainframe
 files 3-36
 format and conversion rules 2-18
 format relationship rules 2-20
 portability 3-2
 program load 2-55

MASK parameter 8-13

Masks 8-13, 8-15, 8-16
 hexadecimal 8-14
 patterns 8-16

Match/merge operations 3-23

MATCHED test 3-25

MEND macro statement 10-5, 10-7

Message levels 8-28, 8-37

MESSAGE statement 8-2, 8-29, 8-36

Messages
 programmer-issued 8-29
 sending 8-28, 8-36
 system-issued 8-29

MIXED
 data format 2-20
 format literals 2-22

MODIFIED testing 8-38

MODIFY file exit, FILE EXIT linkage 2-69

MODIFY subparameter 2-63

MOVE LIKE statement 2-32, 2-42, 5-15

MOVE statement 2-32, 2-41, 5-15

MSTART macro statement 10-7

Multiple dimension arrays 2-46

MVS printer 7-53

N

Native SQL processing 4-25
 reassigning departments 4-28
 retrieving all columns 4-28
 supported commands 4-26
 unsupported commands 4-27
 updating phone numbers 4-29

Newline character
 in UNIX files 3-37

NOACTIVITY parameter 2-26

NOTERMINAL parameter 2-26

Null, definition of 4-8

Numbers
 binary 2-7, 2-8
 integer 2-8
 packed decimal 2-7, 2-8
 unsigned packed decimal 2-8
 zoned decimal 2-7, 2-8

Numeric editing 8-15

Numeric literals
 graph values 9-3

O

Operators, arithmetic 2-32

Output mode 3-7

P

Packed decimal fields 2-78

Packed decimal numbers 2-7, 2-8

Page footers 7-41

PAGE-COUNT field 7-8

PAGE-NUMBER field 7-8

Parameter lists 2-59
 exit 2-62

Parameters, macro 10-4

Parent programs 2-72

Parentheses, in arithmetic expressions 2-33

PARM statement
 SQL parameters 4-4, 4-5
 WORKFILE parameter 2-79

PASCAL calling convention 2-60

PATTERN parameter 2-13, 8-16
 valid characters 8-17

PCB and PSB processing 6-3

Pie chart example 9-5

POINT statement 3-23

Portability between mainframe and workstation 3-2

PRINT statement 2-7, 7-1, 7-2

Printed output 3-28

Printer files 3-28

PRINTER parameter 7-43

Procedure

- AFTER-BREAK 7-34
- AFTER-LINE 7-34
- AFTER-SCREEN 8-31
- BEFORE 3-22
- BEFORE-BREAK 7-34
- BEFORE-LINE 7-34
- BEFORE-SCREEN 8-31
- ENDPAGE 7-34
- INITIATION 8-31
- programmer-defined 8-33
- REPORT-INPUT 7-34
- sort 3-21
- START 3-23
- TERMINATION 7-34
- TERMINATION 8-31

Program

- activities 2-23
- activity section 2-3
- development 2-1
- efficiencies 2-77
- flow control 2-22, 2-23
- library section 2-3
- statement order 2-5

PROGRAM activity 2-4

Program name, IDD interface 5-9

Pseudo-conversational CICS programs 8-36

Publications, related 1-4

PUNCH files 3-4, 3-13

PUT statement 3-12, 3-15, 3-19, 5-15

Q

Qualification

- automatic rules 2-9
- group level 5-9

Quantitative fields 2-7

R

READ statement 3-15

Record address 3-5

Record definitions, IDD interface 5-8

Record format 3-4

- on the workstation 3-30

Record key 3-14, 3-17, 3-27

RECORD-COUNT field 3-6, 4-13

RECORD-LENGTH field 3-5, 4-13

REFRESH action 8-30, 8-33, 8-34

Registers 2-57

Relative files 3-17

- adding records to 3-19
- creating 3-19
- deleting a record 3-19
- random input 3-18
- skip-sequential processing 3-18
- updating a record 3-20

REPEAT statement 8-2, 8-10, 8-27

Report footers 7-42

Report procedures 7-33

REPORT procedures

- coding 7-35
- static working storage 7-36

Report processing

- accumulating report data 7-17
- accumulators 7-18
- annotating reports 7-38, 7-40
- basic structure 7-2
- calculating percentages and averages 7-39
- control field values in titles 7-28
- defining a report 7-7
- detail line 7-18
- display data map 7-3
- extended reporting 7-46
- headings 7-11
- label reports 7-6, 7-15
- lines spacing 7-15
- mixing character sets 7-46
- multiple fonts 7-47
- ordering a report 7-7
- overflow of total values 7-29
- overriding title defaults 7-10
- page footers 7-41

- pre-printed forms 7-14
- quantitative fields 7-7
- report definition statements 7-7
- report display facility 7-43
- report footers 7-42
- report formats 7-5
- report line 7-12
- report line special positioning 7-13
- report title 7-9
- routing printer output 7-43
- sequenced reports 7-17
- sorting data 7-17
- standard report format 7-5
- standard reports 7-9
- system-defined fields 7-8
- terminal output 7-44
- total line 7-18
- work file records 7-3

REPORT statement 7-3, 7-4, 7-7

- DTLCTL parameter 7-23
- PRINTER parameter 7-43
- SPREAD parameter 7-15
- SUMCTL parameter 7-24
- SUMFILE parameter 7-32
- SUMSPACE parameter 7-31

REPORT subactivity 2-4

REPORT-INPUT report procedure 7-37

RESHOW action 8-26, 8-33, 8-35

RETAIN option 3-13

RETRIEVE statement, IMS/DLI 6-1, 6-4

Revision summary 1-5

ROLLBACK statement 4-4

Rollbacks 2-25

ROW statement 8-2, 8-27

- ALL parameter 8-11
- ATTR parameter 8-5, 8-8, 8-10
- COL parameter 8-9
- ERROR parameter 8-26, 8-29
- FILL parameter 8-12
- JUSTIFY parameter 8-12
- MASK parameter 8-12, 8-15
- PATTERN parameter 8-16
- UPPERCASE parameter 8-15
- VALUE parameter 8-25

ROWID 4-27

S

S (static) working storage fields 2-7

SBCS, data format 2-20

Scatter diagram example 9-9

SCREEN activity 2-4, 8-1

- terminating 8-36

Screen format

- function key area 8-3
- message area 8-2
- screen items 8-4
- title area 8-2
- work area 8-2

Screen Painter 8-2

Screen procedures 8-31

Screen processing 8-1

- action bars 8-52
- activity flow 2-24
- attributes 2-12, 8-8
- basic structure 8-1
- borders 8-3
- branch actions 8-33
- character set 8-20
- cursor placement 8-26
- decimal alignment 8-16
- declared attributes 8-5
- determining cursor location 8-38
- dialog boxes 8-51
- displaying arrays 8-27
- dynamic screen attributes 8-46
- edit masks 8-14
- edit masks for display 8-13
- error conditions 8-25
- field attributes 8-10
- field-specific help screens 8-50
- fill characters 8-12
- formatting item for display 8-11
- function key area 8-29
- function keys 8-6, 8-30
- HEX format display 8-14
- input data edit 8-14
- instream table example 8-46
- justifying field contents 8-13
- message area 8-28
- message attributes 8-29
- null entry field 8-12
- overriding default justification 8-13
- pattern 8-20
- pop-up windows 8-51

providing help screens 8-48
pull-down menus 8-52
re-displaying 8-35
repeating data rows 8-27
restoring screen 8-34
rows 8-9
sample applications 8-45
screen format 8-2
setting errors procedure 8-39
state tables 8-22
subscripts 8-27
system-defined fields 8-6
terminal keys 8-6, 8-30
title area 8-7
title attributes 8-8
two-dimensional arrays 8-28
underscore entry field 8-12
uppercase convert 8-15
using a menu 8-47
value checking 8-25
work area 8-9

SCREEN statement 8-2, 8-4
 LINESIZE parameter 8-7

SEARCH statement 2-43, 2-44

Security access code for macros 10-3

Segmented data 5-9

SELECT statement 3-22, 4-17, 4-18, 4-24

SEQUENCE statement 7-7, 7-17, 7-37, 9-3

Sequential access mode 3-8

Sequential files
 automatic processing 3-11
 CARD input 3-11
 controlled processing 3-11
 fixed length 3-12
 variable-length 3-12, 3-13

SEQUENTIAL files 3-10

SET statement 8-2, 8-10, 8-15, 8-27, 8-29, 8-39
 example 8-46

Shift codes 2-22

Signed (quantitative) fields 2-7

Single dimension arrays 2-45

Single file keyed processing 3-22, 3-27

Skip-sequential processing, relative files 3-18

SORT activity 2-4

Sort flow 2-25

Sort procedures 3-21

SORT statement 3-20

Sorting files 3-20

Spooling subsystem 3-28

SQL data types 4-9

SQL databases
 accessing data 4-18
 automatic cursor management 4-2
 automatic processing 4-18
 automatic retrieval without file 4-3
 CA-Easytrieve SQL files 4-17
 catalog INCLUDE facility 4-8
 COMMMIT command 2-28, 2-29
 communications area fields (SQLCA) 4-13, 4-25
 controlled processing 4-20
 deleting from SQL file 4-23
 inserting SQL row 4-23
 joining tables 4-20
 library section definition 4-7
 native processing 4-25
 NULLable field processing 4-19
 NULLable fields 4-8
 overriding default SELECT statement 4-19
 PARM statement parameters 4-4
 programming methods 4-2
 random processing 4-21
 retrieval without a file 4-23
 ROLLBACK command 2-28, 2-29
 syntax checking 4-12
 system-defined fields 4-13
 updating CA-Easytrieve SQL files 4-22

SQL DECLARE statement 4-25

SQL INCLUDE statement 4-8, 4-17, 4-23, 4-26

SQL SET CURRENT SQLID statement 4-5

SQL statement rules 4-3

SQLCODE field 4-18, 4-24, 4-26

SQLSYNTAX, in PARM statement 1-9

Stack usage 2-61

START procedure 3-23

Statement order 2-5

STATUS parameter 3-5

STOP EXECUTE statement 7-4

Storage management

- mainframe 2-56
- workstation 2-60, 2-66
- Structured programming 2-1
- Subject element 2-15, 2-16, 2-18, 2-20
- subprogram linkage
 - Assembler 2-57
- Subprogram linkage 2-13, 2-57
 - COBOL 2-58
- Subscripting 2-44
 - one-dimensional arrays 2-49
 - three-dimensional arrays 2-50
 - two-dimensional arrays 2-49
- Subscripts 2-10, 2-78
 - system-defined 8-27
- SUM statement 7-7
 - label reports 7-15
- SUMCTL parameter 7-24
 - DTLCOPY subparameter 7-27
 - DTLCOPYALL subparameter 7-28
 - TAG subparameter 7-26
- SUMFILE data 7-18
- SUMFILE parameter 7-32
- SUMSPACE parameter 7-31
- Synchronized File Processing (SFP) 3-22
 - conditional expressions 3-26
 - input process 3-24
 - record availability 3-24
 - special IF statements 3-25
 - updating a master file 3-27
- Syntax conventions 1-5
- SYSPRINT 3-29, 7-43, 7-44
- System-defined file fields 3-5
- SYSUSERID field 8-7

T

- Table processing 2-42, 2-78
 - argument 2-42
 - defining tables 2-42
 - external tables 2-42
 - instream tables 2-42, 2-43
 - search arguments 2-43
 - SEARCH statement 2-43, 2-44

- Tables and arrays, IDD interface 5-9
- TAG subparameter 7-26
- TALLY field 7-8, 7-19, 7-40
- Target programs 2-75
- TERM-COLUMNS field 8-6
- Terminal ID 3-28
- TERMINAL parameter 2-26
- TERMINATION report procedure 7-42
- TERMINATION screen procedure 8-31, 8-33
- TERM-NAME field 8-7
- TERM-ROWS field 8-7
- TITLE statement 7-9, 8-2, 9-2, 9-4
 - COL parameter 8-7
- TRANSFER statement 2-55, 2-75
 - implementation 2-76
 - workstation 2-77
- TRANSFER Statement
 - CA-Easytrieve/ESP interactive execution 2-77
- Truncation 2-34

U

- Unit of work 2-25
 - SQL 4-4
- UNIX files 3-37
 - newline character 3-37
- Unsigned (non-quantitative) fields 2-8
- Unsigned packed decimal numbers 2-8
- Update mode 3-8
- USING parameter 2-59, 2-71, 2-74, 2-75

V

- VALUE parameter 8-25
- VALUE statement 9-2, 9-4
- Varying length fields 2-10
 - assigning and moving 2-11
 - comparing 2-12

- rules 2-40
- Vertical axis on graph 9-4
- Vertical bar chart example 9-6
- Virtual File Manager (VFM) 2-79, 3-13, 7-4, 7-44
- VSAM ESDS files 3-10
- VSAM files 3-4
 - creation 3-12
 - KSDS files 3-14
- VSAM password protection 10-3
- VSAM RRDS 3-17
- VSE printer 7-54

W

- W (work) working storage fields 2-6
- WARNING message level 8-28
- WORKAREA parameter 3-3
- WORKFILE parameter, PARM statement 2-79
- Working storage fields 2-6, 2-7
- Workstation
 - allowed field types 3-31
 - bounds checking 2-45, 2-79
 - CALL statement 2-60
 - CARD files 3-11
 - compiler switches 2-76
 - compiler switches 2-73
 - data types supported (SQL) 4-30
 - error condition handling 2-65, 2-70
 - error conditions 8-26

- executing on 2-78
- exit parameter list 2-62
- FABS ISAM files 3-14
- file code system 3-31
- file exits 2-54, 2-59
- FILE statement 3-29
- file types 3-30
- files 3-29
 - fixed length relative files 3-17
- format and conversion rules 2-14
- format relationship rules 2-15
- HLLAPI 2-74
- host interface 2-74
- invoking subprograms in other languages 2-62
- LINK statement 2-73
- linkage conventions 2-60, 2-66
- Local Area Networks (LANs) 3-10
- logical record length 3-31
- MODIFY exits 2-63
- program linking 2-60, 2-65
- PROGRAM statement 2-76
- record format 3-30
- Relative files 3-17
- SEQUENTIAL files 3-10
- sort program 3-21
- standard return code convention 2-61
- storage management 2-60, 2-66
- supported file types 3-32
- synchronized file processing 3-22
- TRANSFER statement 2-76

WRITE statement 3-15, 3-16, 3-17, 5-15

Z

- Zoned decimal numbers 2-7, 2-8