Advantage™ CA-Easytrieve®

Introduction to the Language Guide

Contents

Chapter 1: Overview

Chapter 2: Programming with CA-Easytrieve

Chapter 3: Library Section - Describing and Defining Data

Chapter 4: Activity Section - Processing and Logic

Chapter 5: Activity Section - Input and Output

Chapter 6: Activity Section - Reporting

Chapter 7: Activity Section - Screens

Chapter 8: Activity Section - Graphs

Chapter 9: System-Defined Fields

Index

Chapter

1

Overview

Introduction

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

CA-Easytrieve operates in batch and online modes 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 or routed to a printer.

CA-Easytrieve/Workstation operates on the IBM/PC (or 100 percent 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.

Program Examples

Most program examples in this guide use an input file named PERSNL. This file is made available when CA-Easytrieve is installed so that new users can type in and execute program examples exhibited in the documentation. Some program example output, such as reports, have been edited or shortened for illustrative purposes. Reports you produce from the PERSNL file can be much longer than the ones shown in this guide.

About This Guide

The purpose of this guide is to teach new programmers of CA-Easytrieve how to write CA-Easytrieve programs. It is not intended as a reference guide. Not all features of CA-Easytrieve are described here, nor are all features described available in all implementations of CA-Easytrieve.

This guide assumes that you have some familiarity with data processing concepts and that you have used a computer before. It is *not* required that you have programmed before in other languages. This guide also assumes you are either familiar with the operating environment at your site or you have access to people who can help you. Once you know how to type in and execute programs at your site, this guide teaches you what you need to know about CA-Easytrieve.

By the time you finish the "Programming with CA-Easytrieve" chapter, you will be able to write standard reports, screen transactions, and simple graphs with CA-Easytrieve. For many of you, this may be all you want to do. For the rest of you, there is much, much more.

Because CA-Easytrieve is a compiled language that runs in a multitude of data processing environments, the examples in this guide are generic and do not take into account variations between different installations. It would be impossible to address the specifics of all the operating environments CA-Easytrieve can run under.

Reading The Guide

This guide is designed to be read in three passes or readings. Each successive reading takes you through slightly more advanced and/or specialized material. This helps less experienced users of CA-Easytrieve stay interested while gradually building up their knowledge of the topics presented.

Reading One

This chapter and the tutorial in the "Programming with CA-Easytrieve" chapter are meant to be read in their entirety by all users. The tutorial permits those of you who have an interest in more detail to branch off to first level readings of appropriate chapters at various intervals. You are always directed to return to the tutorial after you have completed the material at the end of such a branch. The goal for the first pass through the guide is to read this chapter and the "Programming with CA-Easytrieve" chapters and the first level readings of all the rest of the chapters. This gives you a good understanding of CA-Easytrieve basics and enables you to perform the following tasks:

- Write a complete CA-Easytrieve program using automatic input and output features.

- Generate standard reports, screens, and graphs.
- Perform calculations and use conditional expressions.
- Be familiar with the system-defined fields provided with CA-Easytrieve.

Reading Two

The second reading begins at level 2 of Chapter 3 and continues with level 2 of Chapters 4 through 8. After completing this second reading, you should be able to perform these tasks:

- Write slightly more complex CA-Easytrieve reports using programmer controlled input and output commands.
- Generate label reports.
- Perform data assignments and moves as well as use loops and branching in program logic.
- Perform basic debugging techniques.
- Write slightly more complex screen activities for your online applications using branch actions and SCREEN procedures.
- Write slightly more complex graph activities, including specifying graph headings and summing graph values.

Reading Three

The third and final reading continues your journey through Chapters 3 through 8. It teaches you more sophisticated commands and techniques available with CA-Easytrieve. After this reading you should be able to perform these tasks:

- Use advanced FILE statement parameters including VIRTUAL and EXIT.
- Perform sorts.
- Use procedures and tables.
- Perform programmer controlled input and output of randomly accessed files including VSAM.
- Use REPORT procedures.
- Override system-defined options for your screen activities.
- Determine cursor location and test for field modification on a screen.
- Sequence a graph and determine the display resolution of a graph.

Organization

This guide is divided into nine chapters:

Chapter 1, Overview Introduces you to CA-Easytrieve and its capabilities. It also describes the basic structure of a CA-Easytrieve program.

Chapter 2, Programming with CA-Easytrieve Takes you, tutorial style, through the process of creating a CA-Easytrieve report, a SCREEN activity, and a GRAPH subactivity.

Chapter 3, Library Section - Describing and Defining Data Tells you how to describe the files you use for processing. It includes special features that CA-Easytrieve provides to make this job easier.

Chapter 4, Activity Section - Processing and Logic Teaches you the basics of writing an application program with CA-Easytrieve. Everything from IF statements to table processing.

Chapter 5, Activity Section - Input and Output Teaches you how to handle (or let CA-Easytrieve handle) input and output files, from automatic I/O to random access of indexed files.

Chapter 6, Activity Section - Reporting Shows you all the things you did not learn about reporting in Chapter 2, "Programming with CA-Easytrieve".

Chapter 7, Activity Section - Screens Shows you all the things you did not learn about screen transactions in Chapter 2, "Programming with CA-Easytrieve".

Chapter 8, Activity Section - Graphs Shows you all the things you did not learn about graph processing in Chapter 2, "Programming with CA-Easytrieve".

Chapter 9, System-Defined Fields Discusses the four types of system-defined fields available in CA-Easytrieve, "Programming with CA-Easytrieve".

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

Other CA-Easytrieve Publications

In addition to this guide, Computer Associates provides the following CA-Easytrieve documentation:

Name	Contents
CA-Easytrieve/Online User Guide	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.
CA-Easytrieve/Online Installation Guide	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.
CA-Easytrieve/Online CA-Activator Supplement	How to install and maintain CA-Easytrieve/Online on your MVS system, using the Computer Associates SMP/E software interface called CA-Activator.
CA-Easytrieve/Online Administrator Guide	Provides system administrators with the information needed to set up, customize, and administer a CA-Easytrieve/Online system.
CA-Easytrieve UNIX User Guide	Helps compile, link-edit, and execute CA-Easytrieve programs in the UNIX environment from the operating system command line.
CA-Easytrieve Programmer Guide	How to create efficient CA-Easytrieve programs and how to analyze and modify existing CA-Easytrieve programs. How to apply these programs to various application tasks.

Continued

Continued

Name	Contents
Ca-Easytrieve Language Reference Guide	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.

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-Pan/SQL SQL Interface Installation Guide*

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.

Continued

Continued

Notation	Meaning
BOLD	Bold text in program code is used to highlight an example of the use of a statement.
CAPS	All capital letters indicate a CA-Easytrieve keyword, or within text descriptions, indicate a name or field used in a program example.
lowercase italics	Lowercase italics represent variable information in statement syntax.

Capabilities

CA-Easytrieve has the capabilities of a retrieval system as well as the comprehensiveness and flexibility required for complex reports, data extraction, and file maintenance requirements.

File Access

The file access features of CA-Easytrieve provide all standard retrieval system capabilities, plus the following:

- Accepts any number of input or output files (constrained by memory).
- Both forward and backward browses of files.
- Synchronizes file processing (based on keys) of an unlimited number of files, including matched conditions and duplicate checking. This reduces complex matching logic to a single statement.
- Tests for file availability and current record count.
- Provides in-core binary search of external or instream table files.
- Prints a file status and error analysis report at point of error during abnormal termination.
- Provides an easy method for establishing temporary work files without special job control or file allocation statements.

Field Definition

The methods CA-Easytrieve uses to define all types of record structures and field formats are consistent and easy to use, including:

- Defining all field formats, including binary and unsigned packed fields.
- Providing flexible edit masks for report formats or displaying data, including blank-when-zero and hexadecimal display.
- Establishing initial values for working storage fields.
- Providing default report headings to enhance standards.
- Allowing multiple use of field definitions with the COPY keyword, reducing coding and maintenance.

Logic Processing

The purpose of any information retrieval and application development system is to provide complete conditional logic. CA-Easytrieve provides this logic, plus the following:

- Provides standard programming constructions such as nested IF, DO, and PERFORM statements.
- Provides powerful calculation capabilities, including bit manipulation.
- Performs special tests useful in editing, including alphabetic, numeric, spaces, zero, null, and test under mask.
- Allows string manipulation.
- Supports move for corresponding fields.
- Includes special one-time procedures for start of processing and termination of processing.
- Sorts on any number of keys.

File Output

Routine file maintenance is faster and simpler because of the enhanced capabilities of CA-Easytrieve, including:

- Loading and updating files, including VSAM, SQL, and other popular database management systems.
- Saving report extract work files for subsequent use.
- Providing a selective hexadecimal dump of a file or specific fields.

SQL Processing

CA-Easytrieve provides complete facilities for processing SQL databases, including:

- A full set of native SQL statements.
- Comprehensive language extensions that provide full management of SQL cursors.

Report Output

The reporting features of CA-Easytrieve make producing reports a simple, uncomplicated process. The flexibility built into the system through specialized report procedures makes it easy to produce customized reports without compromise. CA-Easytrieve:

- Produces unlimited reports from a single pass of the data.
- Automatically formats reports.
- Provides customizing alternatives to all report format features.

- Provides mailing labels of any size.
- Provides control breaks on any number of keys.
- Automatically creates a summary file containing subtotals.
- Processes only those fields that are required by your REPORT statements.
- Generates reports to separate logical printers or other output media. You can also send reports to the Report Display Facility so they can be browsed and, optionally, printed.
- Provides control break level access for special logic processing, which is useful when only certain report lines are to be generated for certain specific levels of control breaks.
- Provides specialized report procedures for user flexibility (BEFORE/AFTER-LINE, ENDPAGE, TERMINATION, BEFORE/AFTER-BREAK, REPORT-INPUT).
- Permits explicit positioning of print layout for pre-printed forms.

Screen Processing

CA-Easytrieve also provides the ability to create screen transaction programs with ease. Just as CA-Easytrieve removes the tedious details from report writing, its declarative structure removes the mundane tasks associated with screen processing. CA-Easytrieve provides the following:

- Ability to produce multiple screens from a single program.
- Automatically centered screen titles.
- Automatically validated terminal keys.
- An automatically created IBM SAA function key display area.
- Automatically edited screen data.
- Ability to send programmer-issued messages using one of three SAA severity levels.
- Special-named screen procedures for user flexibility, such as: BEFORE-SCREEN, AFTER-SCREEN, INITIATION, and TERMINATION.
- Ability to test for field modifications and cursor placement.
- Support for pop-up window screens.

Graph Processing

Additionally, CA-Easytrieve provides the ability to produce bit-mapped presentation graphs with a non-procedural technique very similar to reporting. Graph processing includes the following:

- Ability to produce multiple graphs from a single program.
- Ability to produce pie charts, bar charts, line graphs, and scatter diagrams.
- Automatically centered graph titles.
- System-defined function key assignments to request help, exit, or print.
- Ability to optionally summarize y-axis data points and categorize by x-axis data points.
- Ability to optionally sequence the graph values.

Virtual File Manager

VFM provides an easy method for establishing temporary work files without special job control or file allocation statements. By using VFM, you can establish your own extract or temporary files using only CA-Easytrieve keywords. VFM's own data management techniques ensure its operating efficiency standards, including:

- Maintaining more information in memory. If the memory area is exhausted, VFM writes the excess data to a single spill area.
- Defining only one physical file.
- Determining the best blocking factor based on device type, providing a 90 percent disk utilization.
- Releasing and recovering occupied space as the virtual file is read back into your program.
- Automatically spooling files created as a result of sequenced reports or multiple reports in the same activity.

Debugging Capabilities

The debugging aids provided by CA-Easytrieve ensure that all information necessary to identify the cause of an abnormal termination is easily readable by:

- Providing an error analysis report which pinpoints most errors immediately including the source statement number in error and/or a FLOW table of what statements were executed in what series.
- Providing optional data processing oriented displays such as DMAPs and PMAPs.
- Trapping invalid file references and checking field boundaries during execution to prevent a system dump.

Current Technology

CA-Easytrieve represents the maximum in efficiency because it has been developed with the latest in programming technology, including:

- Mapping programs in 4K segments whenever possible.
- Mapping working storage on double word boundaries.
- Providing a one-pass compiler.
- Directly generating the object code.
- Providing security on VSAM and SQL usage.

Structure of a CA-Easytrieve Program

Before beginning the tutorial in Chapter 2, "Programming with CA-Easytrieve," it is helpful to have a basic understanding of how a CA-Easytrieve program is structured.

Each CA-Easytrieve program can contain an *environment* section, a *library definition* section, and one or more *activity* sections. The environment and library definition sections are optional but at least one activity section is required.

```
                ⎛
                ⎜ Environment section   (optional)
CA-Easytrieve   ⎜
   PROGRAM      ⎨ Library section       (optional)
                ⎜
                ⎜ Activity section(s)   (one required)
                ⎝
```

Environment Section

The environment section establishes parameters for the program. This section permits you to override standard CA-Easytrieve options and to choose a mode of operation. The environment section is not required for most of the examples in this guide. For a complete discussion of the environment section, see the PARM Statement in the *CA-Easytrieve Language Reference Guide*.

Library Definition Section

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

Activity Section

The activity section is the only mandatory 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 a screen-oriented transaction. 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 printed reports.
- 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 and are discussed in Chapter 4, "Activity Section - Processing and Logic."

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

Environment Section

```
PARM ...
```

Library Section

```
FILE ...
  DEFINE ...
...
```

Activity Section

```
PROGRAM ...
  (statements)
  (program procedures)
JOB ...
  (statements)
  (job procedures)
  REPORT ...
    (report procedures)
  GRAPH
SORT ...
  (sort procedures)
SCREEN ...
  (screen procedures)
...
```

Sample Program

The following shows an example of a simple CA-Easytrieve program. This program produces a standard report that is used in the next section as a starting point for the tutorial. We show it here to further illustrate the basic structure of a CA-Easytrieve program. (The environment section is omitted.)

```
FILE PERSNL FB(150 1800)                    ┐
  EMPNAME  17  8  A                         │
  EMP#      9  5  N                         ├ Library Section
  DEPT     98  3  N                         │
  GROSS    94  4  P  2                      ┘

JOB INPUT PERSNL NAME FIRST-PROGRAM         ┐
  PRINT PAY-RPT                             │
REPORT PAY-RPT LINESIZE 80                  ├ Activity Section
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'     │
  LINE 01 DEPT EMPNAME EMP# GROSS           ┘
```

The above program produces the following output. As is the case in many of the output examples in this guide, it has been edited for illustrative purposes.

```
01/31/91          PERSONNEL REPORT EXAMPLE-1        PAGE   1


              DEPT    EMPNAME   EMP#      GROSS

              903     WIMN      12267     373.60
              943     BERG      11473     759.20
              915     CORNING   02688     146.16
              935     NAGLE     00370     554.40
              911     ARNOLD    01963     445.50
              914     MANHART   11602     344.80
              917     TALL      11931     492.26
              918     BRANDOW   02200     804.64
              911     LARSON    11357     283.92
              932     BYER      11467     396.68
              921     HUSS      11376     360.80
              911     POWELL    11710     243.20
              943     MCMAHON   04234     386.40
```

If you are familiar with other programming languages or report generators, you realize that CA-Easytrieve takes care of a lot of details for you; details that you would otherwise spend countless hours controlling and typing in yourself. In the next chapter, you are given a chance to jump right in and start creating your own CA-Easytrieve programs.

We hope you enjoy programming with CA-Easytrieve.

Chapter 2

Programming with CA-Easytrieve

Introduction

You need information and you need it now. The data processing department is backlogged two weeks. You know where your data is and you have been given access to a programming language called CA-Easytrieve. What do you do? That's what this tutorial is about. In this tutorial, you learn:

- How to create a report quickly using CA-Easytrieve
- How to redefine your report to suit your needs
- How to create a simple SCREEN activity
- How to create a simple GRAPH subactivity.

Reading This Tutorial

This tutorial is divided into eight lessons. At the end of each lesson, you are given an opportunity to branch off and explore further details of the material just covered. Or, you can read the tutorial straight through, catching up on the details later. Either way you do it, when you are finished you'll be able to write a CA-Easytrieve program.

Tutorial Lessons

The eight lessons in this chapter correspond (generally) to Chapters 3 through 8 of this guide. The topics covered are:

Lesson 1: CA-Easytrieve library section including FILE and DEFINE statements.

Lesson 2: CA-Easytrieve activity section including JOB and IF statements. Also a look at working storage fields defined in the library section.

Lesson 3: CA-Easytrieve activity section including report output with the PRINT Statement. Also, a return to the library section for a look at the HEADING and MASK parameters of the DEFINE statement.

Lesson 4: CA-Easytrieve activity section including the REPORT statement and report definition statements.

Lesson 5: Using a CA-Easytrieve SCREEN activity to create an online adding machine.

Lesson 6: Improving the readability of your screen with the ROW statement.

Lesson 7: Adding error trapping to your SCREEN activity.

Lesson 8: Using a CA-Easytrieve GRAPH subactivity to create a simple scatter diagram.

Lesson 1

To start out, we're going to show you a CA-Easytrieve program. This program is completely executable; so if you are at a terminal, feel free to type it in and run it. However, you are not required to have access to a terminal to follow the lessons presented in this tutorial.

```
FILE PERSNL FB(150 1800)
  EMPNAME  17  8  A
  EMP#      9  5  N
  DEPT     98  3  N
  GROSS    94  4  P  2

JOB INPUT PERSNL NAME FIRST-PROGRAM
  PRINT PAY-RPT
REPORT PAY-RPT LINESIZE 80
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE 01 DEPT EMPNAME EMP# GROSS
```

This sample program is very short, only 11 lines long. But, packed into the program is the power to produce a completely formatted report including: date, page number, title, column headings, properly spaced detailed lines, and more. In other languages, you might expect to write 20 or more times this much code to produce the same report.

The Report Your Program Creates

The sample program produces a report similar to the following:

```
01/31/91         PERSONNEL REPORT EXAMPLE-1        PAGE    1


               DEPT    EMPNAME   EMP#      GROSS

               903     WIMN      12267     373.60
               943     BERG      11473     759.20
               915     CORNING   02688     146.16
               935     NAGLE     00370     554.40
```

```
911     ARNOLD    01963     445.50
914     MANHART   11602     344.80
917     TALL      11931     492.26
918     BRANDOW   02200     804.64
911     LARSON    11357     283.92
932     BYER      11467     396.68
921     HUSS      11376     360.80
911     POWELL    11710     243.20
943     MCMAHON   04234     386.40
```

This report is simply an edited display of fields from an employee file named PERSNL. (As we mentioned in the previous chapter, this sample file is provided with CA-Easytrieve. Ask your system administrator where it is stored at your site.) It's a good starting point for describing some of the most important CA-Easytrieve keywords.

One Statement at a Time

Let's go on by examining the cause and effect relationship between our program and its report, one statement at a time. We'll start out with the CA-Easytrieve library definition section (described in Chapter 1, "Overview").

We're going to keep it simple; if you become impatient and feel you want more, remember you'll be directed to more detailed information at the end of this lesson.

The FILE Statement

The first line of our program looks like this:

```
FILE  PERSNL  FB(150 1800)
```

This line contains the CA-Easytrieve FILE statement. A FILE statement must be included for every file you use as input to or output from your program. It tells CA-Easytrieve where to get the data you want processed and can also tell it some things about how that data is stored. To do this, it must include a file name, and in our example, that name is PERSNL.

The rest of line 1 is optional. It tells CA-Easytrieve some information about how the PERSNL file is stored, which makes accessing it more economical. The PERSNL file contains records of a fixed length of 150 characters stored in 1800 character blocks. This is indicated as one parameter, FB(150 1800). (FB stands for Fixed, Blocked.) Multiple subparameters are always enclosed by parentheses in CA-Easytrieve. Since record length (150) and blocksize (1800) are mandatory subparameters of FB, we included them in parentheses.

The DEFINE Statement

There are four DEFINE statements in our program:

```
EMPNAME   17    8    A
EMP#       9    5    N
DEPT      98    3    N
GROSS     94    4    P    2
```

These four lines describe fields in a record of the PERSNL file. You don't see the word DEFINE in the above lines, but it is implied. If we wanted, we could have written these lines as:

```
DEFINE    EMPNAME  17    8    A
DEFINE    EMP#      9    5    N
DEFINE    DEPT     98    3    N
DEFINE    GROSS    94    4    P    2
```

The DEFINE statement can also be used right in the middle of your program logic if you need a quick working storage field. Used there, the DEFINE keyword is required; it cannot be implied. We'll cover this more thoroughly in Chapter 3, "Library Section - Describing and Defining. Data."

The DEFINE statements just shown describe four of the fields in a record of the PERSNL file. They don't have to describe all the fields in the record, or the spaces between fields, because in CA-Easytrieve that isn't necessary. You describe only what you need to use.

The basic components of a field definition are fairly easy to understand; let's label them for you.

Field Name	Starting Position in Record	Length of Field	Data Type	Number of Decimal Positions
EMPNAME	17	8	A	
EMP#	9	5	N	
DEPT	98	3	N	
GROSS	94	4	P	2

When describing a field, you need to identify its name, give its location in the record (starting position), its length, its type, and number of digits to the right of the decimal point if any.

These items must be identified in the order shown above (left to right) and must be separated by spaces. In our example, we line them up in columns (top to bottom) for readability but that's not required.

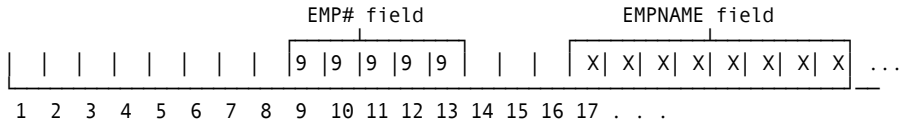
Field Name

The field name identifies the field as a unique storage location and is what you use later to refer to your data.

Starting Position

The starting position, or location, is where (in the record) the first character of the field begins.

Beginning at the first character of data in a record of the PERSNL file, you count nine characters to the right to land on the first character of the EMP# field:



A Physical record in the PERSNL File

By counting 17 characters to the right from position 1, you land on the first character of the EMPNAME field and so on.

Fields do not have to be described in your program in the same order that they occur in the record. In our example, we define EMPNAME before EMP# even though it physically comes after EMP# in the record.

Field Length

The length of the field is simply the number of characters of storage space (bytes) the field takes up in the record. As illustrated above, you can see that EMP#, which has a field length of 5, takes up 5 character positions, or *bytes,* in the PERSNL record.

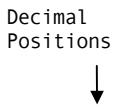
Data Type

Data type describes the kind of data stored in a field. In the example we are following, the fields consist of three different data types.

Field	Data Type	Purpose
EMPNAME	A - Alphanumeric	Stores non-numeric data
EMP#	N - Numeric	Stores numbers in zoned decimal format
DEPT	N - Numeric	Stores numbers in zoned decimal format
GROSS	P - Packed decimal	Stores numbers in internal packed decimal format

Decimal Positions

In our example, the field GROSS is the only field that contains numbers to the right of a decimal point:



```
GROSS  94   4   P    2
```

When this field is printed on your report, it shows up with two numbers to the right of a decimal point (for example: 999.99).

Reviewing the Library Section

The statements you have covered so far (FILE and DEFINE) comprise the bulk of the CA-Easytrieve library section. These two statements define the *library* of data CA-Easytrieve uses as input to any processing activities.

- FILE tells CA-Easytrieve about the data file you are accessing (or creating).
- DEFINE tells CA-Easytrieve which fields you'll be using from the file.

Once you have defined your data, you can go on to processing activities.

For More Information

To add detail to your understanding of the library section, turn to Chapter 3, "Library Section - Describing and Defining Data".

If you are content with what you have learned so far, proceed to the next lesson.

Lesson 2

In the previous lesson of this tutorial, we showed you a complete CA-Easytrieve program and discussed part of it called the library definition chapter. Yet, we haven't explained how your data is processed to produce a report.

In this lesson, we'll discuss the JOB activity, which defines and initiates all processing activities in our sample program. Also, we're going to add conditional statements and a calculation for salary deductions.

First, let's talk about the JOB statement.

The JOB Statement

The first line after the library section in our sample program is prefixed by the word JOB:

```
JOB INPUT PERSNL NAME FIRST-PROGRAM
```

When CA-Easytrieve encounters a JOB statement, it knows that it is about to begin some form of processing. The JOB statement can also automatically provide input (if input is available) to the processing statements that follow it.

In the line shown above, taken from our sample program, everything but the word JOB is optional. Let's explain.

Input to a JOB Activity

The word JOB, in CA-Easytrieve, is like a sign that reads "Work in Progress." It indicates that processing is to follow. Typically, processing requires some kind of file input.

Most programming languages require you, the user, to control the availability of input files. Usually, files are opened and then some sort of input statement is executed in a loop, checking for an end-of-file condition each time it is executed.

Although CA-Easytrieve gives you the flexibility to control input, it also has the power to do all the "dirty work" for you. This is called *Automatic* Input.

Automatic Input

By using the INPUT parameter of the JOB statement, you indicate that the named file (in this case PERSNL) be made *automatically* available to your program. It's like saying, "I want to use this file, please." And letting CA-Easytrieve do the rest.

CA-Easytrieve is smart; if you don't specify INPUT, it looks for input and uses the first file described in the library section. (Unless the JOB activity is preceded by a SORT activity, in which case CA-Easytrieve uses the output from that SORT. See Chapter 5, "Activity Section - Input and Output" for more information on SORT.) Since our sample program has only one input file (PERSNL), the INPUT parameter on the JOB statement is completely optional. Without it, CA-Easytrieve looks for input and uses the first file (the only file) in our library section, PERSNL.

Naming a JOB Activity

The next thing after the INPUT parameter in our sample program is the word NAME. This simply tells CA-Easytrieve that a job name follows.

You normally name a JOB activity for documentation purposes only. It helps to give JOB activities a descriptive name especially when you have more than one in your program. In our example, we named the JOB activity FIRST-PROGRAM. We did this by typing the parameter NAME followed by a name of our choice.

A Look at Logic

The program we have discussed so far is capable of producing a complete report. All that this program requires is a description of the data we want to print and a few other lines of code we'll discuss in the following pages of this tutorial.

In the mean time, to make things a little more interesting, we're going to add a few things to our program.

A New Condition

We've been talking a lot about a report program that simply extracts some data from a file and prints it out. Granted, CA-Easytrieve makes doing this very easy and automatic, but it is capable of doing so much more!

If This, Then That

Let's assume your boss just came in and said that the report you're working on has to include net pay and deductions. Let's look again at the program we've been working on so far.

```
FILE PERSNL FB(150 1800)
  EMPNAME  17  8  A
  EMP#      9  5  N
  DEPT     98  3  N
  GROSS    94  4  P  2

JOB INPUT PERSNL NAME FIRST-PROGRAM
  PRINT PAY-RPT
REPORT PAY-RPT LINESIZE 80
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE 01 DEPT EMPNAME EMP# GROSS
```

The program accesses a field called GROSS which contains employee gross pay. Since net pay (take home pay) is the gross pay minus any deductions, you quickly realize that you need to figure out what to deduct. As it happens, you're a payroll expert, and you know that employees who make $500 or more get a 28 percent deduction; the rest don't get any deduction.

The condition we've just described can be stated with a simple conditional expression:

```
IF  GROSS  GE  500
  DEDUCTIONS = .28 * GROSS
  NET-PAY = GROSS - DEDUCTIONS
ELSE
  NET-PAY = GROSS
  DEDUCTIONS = 0
END-IF
```

In this expression, we say, "If the gross pay is greater than or equal to 500, deduct 28 percent to give the net pay. Otherwise, if the gross is less than 500, there are no deductions and net pay is the same as gross." CA-Easytrieve requires an END-IF to complete the expression.

Adding Logic to the JOB Activity

Now that we have a logical statement to describe our condition, we simply type it into our program, placing it in the JOB activity after the JOB statement.

```
FILE PERSNL FB(150 1800)
  EMPNAME  17  8  A
  EMP#      9  5  N
  DEPT     98  3  N
  GROSS    94  4  P  2

JOB INPUT PERSNL NAME FIRST-PROGRAM

  IF  GROSS  GE  500
    DEDUCTIONS = .28 * GROSS
    NET-PAY = GROSS - DEDUCTIONS
  ELSE
    NET-PAY = GROSS
    DEDUCTIONS = 0
  END-IF

  PRINT PAY-RPT
REPORT PAY-RPT LINESIZE 80
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE 01 DEPT EMPNAME EMP# GROSS
```

New Logic

There are several details we still need to take care of. We need a place to store the results for our two new variables, DEDUCTIONS and NET-PAY. They can be stored in a place known as working storage.

CA-Easytrieve Working Storage

Unlike many other languages, CA-Easytrieve makes the definition of working storage fields a breeze. You can place them in the library section of your program, or even right in the activity section before the logic that requires them.

To define a working storage field, you use the same type of attributes used to describe other fields. But, you replace the numeric value which normally describes the start location with the letter W.

```
DEDUCTIONS  W  4  P  2
NET-PAY     W  4  P  2
```

The above fields can be described in words as: working storage fields, 4 characters long, in packed decimal format with 2 decimal places. Let's place these fields in the library section of our program (this enables them to be more easily seen than if they were placed in the activity section).

```
FILE PERSNL FB(150 1800)
  EMPNAME     17  8  A
  EMP#         9  5  N
  DEPT        98  3  N
  GROSS       94  4  P  2
  DEDUCTIONS   W  4  P  2
  NET-PAY      W  4  P  2

JOB INPUT PERSNL NAME FIRST-PROGRAM
```

Working Storage Fields

```
  IF  GROSS  GE  500
    DEDUCTIONS = .28 * GROSS
    NET-PAY = GROSS - DEDUCTIONS
  ELSE
    NET-PAY = GROSS
    DEDUCTIONS = 0
  END-IF

  PRINT PAY-RPT
REPORT PAY-RPT LINESIZE 80
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE 01 DEPT EMPNAME EMP# GROSS
```

So far, we've used some elementary logic, calculated some values, and created a place to store those values.

Review of Job Activities

In this lesson of the tutorial, you've learned how to add conditional logic and calculations to your program in order to compute new information. You've also learned how to store the new information. You know that:

- JOB initiates program processing activities and can also provide automatic file input.
- IF is a conditional expression used to make decisions, based on certain criteria.
- W designates a working storage field on the DEFINE statement.

In the next lesson, we'll talk a little about the LINE statement and the statement that is responsible for initiating the printing of your report, the PRINT statement. We'll also be discussing a couple of parameters that allow you to edit and label your data to make it more meaningful, MASK and HEADING.

For More Information

To add detail to your understanding of the JOB activity section, turn to Chapter 4, "Activity Section - Processing and Logic."

If you want to continue with the tutorial to learn how to print a CA-Easytrieve report, proceed to the next lesson.

If you want to continue the tutorial to learn how to create a CA-Easytrieve SCREEN activity, move on to Lesson 5, "Your First Screen Program."

Lesson 3

The CA-Easytrieve PRINT statement is responsible for activating report statements which result in a printed report.

As we'll discuss in this lesson, a report declaration consists of a series of statements that define the format and content of a report. These statements consist of the REPORT statement, report definition statements, and report procedures. So far, we have seen three such statements in our sample program.

- REPORT
- TITLE
- LINE

In our sample program, the PRINT statement occurs directly after the conditional statements we created in the last lesson of this tutorial.

```
  END-IF

  PRINT PAY-RPT
REPORT  PAY-RPT  LINESIZE 80
```

Once the conditional statements have been executed against a record of the PERSNL file, the PRINT statement tells CA-Easytrieve to execute the report definition statements. In the previous PRINT statement example, these statements are identified by a user-supplied name, PAY-RPT. This name ties the PRINT statement to a specific report of the same name as indicated on the REPORT statement. If the report name is not included, CA-Easytrieve executes the first report in the JOB activity section whether or not it has a name.

Once the report statements are executed, control is returned to the beginning of the JOB activity section where the next record is processed, or end-of-file processing is performed. All output routines, line counts, and page advances, are handled automatically. You simply say PRINT and CA-Easytrieve does the rest.

CA-Easytrieve LINE Statement

The last line in the program we've been showing looks like this:

```
LINE  01  DEPT EMPNAME EMP# GROSS
```

This line of code is responsible for the printing of detail lines on the report. It tells CA-Easytrieve what fields to print and the order in which to print them.

To add DEDUCTIONS and NET-PAY to the report output, all we have to do is make the LINE statement look like this:

```
LINE  01  DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

We simply add the names of our two new fields in whatever order we want them to appear.

With this last change we can run our new and improved program to generate a report. Here is the program with all the changes we've made.

```
FILE PERSNL FB(150 1800)
  EMPNAME    17  8  A
  EMP#        9  5  N
  DEPT       98  3  N
  GROSS      94  4  P  2
  DEDUCTIONS  W  4  P  2
  NET-PAY     W  4  P  2

JOB INPUT PERSNL NAME FIRST-PROGRAM

  IF  GROSS  GE  500
    DEDUCTIONS = .28 * GROSS
    NET-PAY = GROSS - DEDUCTIONS
  ELSE
    NET-PAY = GROSS
    DEDUCTIONS = 0
  END-IF

  PRINT PAY-RPT
REPORT PAY-RPT LINESIZE 80
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE  01  DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

Next is the sample output from the program just shown. As you can see, two new columns of information have been added for NET-PAY and DEDUCTIONS:

```
01/31/91            PERSONNEL REPORT EXAMPLE-1                PAGE    1

        DEPT   EMPNAME    EMP#      GROSS       NET-PAY    DEDUCTIONS

        903    WIMN       12267      373.60       373.60         .00
        943    BERG       11473      759.20       546.63      212.57
        915    CORNING    02688      146.16       146.16         .00
        935    NAGLE      00370      554.40       399.17      155.23
        911    ARNOLD     01963      445.50       445.50         .00
        914    MANHART    11602      344.80       344.80         .00
        917    TALL       11931      492.26       492.26         .00
        918    BRANDOW    02200      804.64       579.35      225.29
        911    LARSON     11357      283.92       283.92         .00
        932    BYER       11467      396.68       396.68         .00
        921    HUSS       11376      360.80       360.80         .00
        911    POWELL     11710      243.20       243.20         .00
        943    MCMAHON    04234      386.40       386.40         .00
```

Editing Your Report Output

In the previous lesson of this tutorial, we added two new values to our report, NET-PAY and DEDUCTIONS. Both of these, along with GROSS, are dollar values. Up until now, dollar values have only printed as ordinary numbers with decimal places. We can edit these values so they print with dollar signs by adding an edit mask to the DEFINE statement.

Edit Masks

An edit mask is a pattern of characters that specifies how numeric data be printed. (Alphanumeric fields cannot be edited.) For example, let's add edit masks to the three currency fields in our example program so they print with dollar signs.

```
GROSS          94   4   P   2   MASK (A '$$,$$9.99')
NET-PAY         W   4   P   2   MASK A
DEDUCTIONS      W   4   P   2   MASK (A BWZ)
```

The first thing you'll notice is a new keyword, MASK. MASK is a parameter of the DEFINE statement that designates an edit mask follows. In the above example, the actual mask consists of the characters: '$$,$$9.99'. Masks are always enclosed in single quotes.

The effect on our report of adding the above masks is as follows:

```
01/31/91              PERSONNEL REPORT EXAMPLE-1              PAGE   1

          DEPT   EMPNAME    EMP#      GROSS       NET-PAY     DEDUCTIONS

          903    WIMN       12267      $373.60      $373.60
          943    BERG       11473      $759.20      $546.63      $212.57
          915    CORNING    02688      $146.16      $146.16
          935    NAGLE      00370      $554.40      $399.17      $155.23
          911    ARNOLD     01963      $445.50      $445.50
          914    MANHART    11602      $344.80      $344.80
          917    TALL       11931      $492.26      $492.26
          918    BRANDOW    02200      $804.64      $579.35      $225.29
          911    LARSON     11357      $283.92      $283.92
          932    BYER       11467      $396.68      $396.68
          921    HUSS       11376      $360.80      $360.80
          911    POWELL     11710      $243.20      $243.20
          943    MCMAHON    04234      $386.40      $386.40
```

Note: Any leading zeros are suppressed and each value has one dollar sign. Any all-zero values in the DEDUCTIONS column are printed as blanks.

The following explanations and rules apply to the edit masks in our example:

- Each digit in a field must be designated in the edit mask. Since a four-byte packed decimal field is capable of containing seven digits, we need to designate seven digits in the mask. This is done with $$$$999.
- Dollar signs ($) in the edit mask indicate that a dollar sign is to be printed prior to the first non-zero digit of the printed field. This is called a floating dollar sign. It means that if one or more high-order zeros are stored in the positions where a dollar sign appears in the mask, they are suppressed and replaced with a single dollar sign. For example:

Mask	Field Value	Resulting Output
'$$,$$9.99'	1234567	$12,345.67
	0123456	$1,234.56
	0012345	$123.45

Mask	Field Value	Resulting Output
	0001234	$12.34
	0000123	$1.23
	0000012	$0.12

As the number of leading zeros increases, the dollar sign automatically floats to the right.

- The digit 9, indicates that any value occurring in that position is printed as a digit. In the above example, all values in the *ones* column or to the right of the decimal are printed as digits, even zeros.
- Commas and decimal points are printed just as indicated. In the above example, you can see that commas are suppressed along with high-order zeros for numbers less than 1000.
- When the same mask is to be used on more than one field, you can avoid coding the mask more than once by naming it, and then specifying only the name on subsequent fields. Names can be any letter from A through Y.

  In our example we named the mask used on the GROSS field A. Then, we specified the letter A on the NET-PAY and DEDUCTIONS fields instead of coding the mask again. Remember, multiple parameters and subparameters are enclosed in parentheses.
- To suppress all-zero values from printing (if you find that desirable) simply code BWZ (blank when zero) after the mask or mask name. Because some employees in our report can have zero deductions, we included BWZ.

Field Headings

So far in our example program, field (or column) headings have come directly from the field names themselves. Automatically, CA-Easytrieve uses field names (specified on the DEFINE statement) as column headings, unless column headings are described separately.

One way to describe alternative column headings is with the HEADING parameter of the DEFINE statement. For example, you can replace the somewhat cryptic heading EMP# with the more readable heading EMPLOYEE NUMBER as follows:

```
EMPNAME 17    8    A
EMP#     9    5    N   HEADING ('EMPLOYEE' 'NUMBER')
DEPT    98    3    N
```

By placing each word in single quotes, you indicate that the heading should be stacked, one word over the other.

The following shows how the new heading prints, once the program is run.

```
01/31/91              PERSONNEL REPORT EXAMPLE-1              PAGE   1
```

```
                    EMPLOYEE
DEPT    EMPNAME      NUMBER       GROSS        NET-PAY      DEDUCTIONS
```

You can include headings on the DEFINE statement for any fields you feel need better identification.

Reviewing PRINT, LINE, MASK, and HEADING

In this lesson of the tutorial, we have discussed the workings of the PRINT statement and also how to use the MASK and HEADING parameters to make your reports more readable. You have learned that:

- PRINT activates a report declaration resulting in a printed report.
- LINE determines which fields are printed on your report and in what order they are printed.
- MASK enables you to change the look of fields on your report.
- HEADING enables you to customize column headings on your report.

In this lesson, we have made some minor changes to our on-going program example. Here is how our program looks:

```
FILE PERSNL FB(150 1800)
  EMPNAME    17  8  A
  EMP#        9  5  N  (HEADING ('EMPLOYEE' 'NUMBER')
  DEPT       98  3  N
  GROSS      94  4  P   2   MASK (A '$$,$$9.99')
  NET-PAY     W  4  P   2   MASK A
  DEDUCTIONS  W  4  P   2   MASK (A BWZ)

JOB INPUT PERSNL NAME FIRST-PROGRAM

  IF  GROSS  GE  500
    DEDUCTIONS = .28 * GROSS
    NET-PAY = GROSS - DEDUCTIONS
  ELSE
    NET-PAY = GROSS
    DEDUCTIONS = 0
  END-IF

  PRINT PAY-RPT
REPORT PAY-RPT LINESIZE 80
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE  01  DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

For More Information

If you'd like to learn more about the PRINT statement, you can turn now to Chapter 5, "Activity Section - Input and Output."

To continue the tutorial and learn about report declarations, proceed to the next lesson.

Lesson 4

The example program we have been discussing currently has only three statements in its report declaration. These statements are REPORT, TITLE, and LINE.

```
REPORT  PAY-RPT  LINESIZE  80
  TITLE  01  'PERSONNEL REPORT EXAMPLE-1'
  LINE   01  DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

In this lesson, we discuss these three statements and are also adding four more statements to our program:

- SEQUENCE
- CONTROL
- SUM
- HEADING

The REPORT Statement

The REPORT Statement must be the first statement in your report declaration. It tells CA-Easytrieve that a report is about to be described and is also used to identify the type of report and its various physical characteristics.

In our sample program we identify the report by name (PAY-RPT) and also specify a LINESIZE of 80; both of these are optional. Since our program has only one report, we could have left the report name off of both the PRINT and the REPORT statements. A linesize of 80 restricts report output to 80 characters per printed line. If you are entering programs as we move along and are reviewing output at your terminal, then 80 characters per line is appropriate. (Since most terminals only display 80 characters on a screen.)

Report Definition Statements

Report definition statements define the contents of a report. We'll discuss these statements in the order they must occur in your report declaration. We'll add new statements to our example program as we go, showing the effects on report output.

There are six report definition statements in CA-Easytrieve. When used, they must occur after the REPORT statement and in a specified order, as follows:

- SEQUENCE
- CONTROL
- SUM
- TITLE

- HEADING
- LINE

A clever CA-Easytrieve user came up with a useful mnemonic device for remembering these statements and their order:

```
Siblings  Can  Sometimes  Tell  Horrible  Lies
E         O    U         I     E         I
Q         N    M         T     A         N
U         T              L     D         E
E         R              E     I
N         O                    N
C         L                    G
E
```

The mnemonic may sound silly, but it's effective. All these statements are discussed briefly on the remaining pages of this lesson. After you understand what each of these statements do, you'll see that the order of these statements is very logical.

The SEQUENCE Statement

The SEQUENCE statement causes your report to be sorted on a specified key in ascending or descending order. Let's sequence our current report example on department in ascending order. That is, let's tell CA-Easytrieve to print out all of our employees in order by department number starting with the lowest department number. (The department number is in the field called DEPT.) All we have to do is place the SEQUENCE statement and the field name DEPT after the REPORT statement:

```
REPORT   PAY-RPT   LINESIZE   80
  SEQUENCE DEPT
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE  01 DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

Ascending order is the default for the SEQUENCE statement. For descending order, enter D after the field name, separated by a space.

When we run our program, here's what we get:

```
01/31/91              PERSONNEL REPORT EXAMPLE-1              PAGE   1

                          EMPLOYEE
      DEPT   EMPNAME       NUMBER       GROSS       NET-PAY     DEDUCTIONS

      901    WALTERS       11211         $424.00       $424.00
      903    WIMN          12267         $373.60       $373.60
      912    LOYAL         04225         $295.20       $295.20
      914    MANHART       11602         $344.80       $344.80
      914    VETTER        01895         $279.36       $279.36
      914    GRECO         07231       $1,004.00       $722.88      $281.12
      914    CROCI         08262         $376.00       $376.00
      914    RYAN          10961         $399.20       $399.20
      915    CORNING       02688         $146.16       $146.16
      917    TALL          11931         $492.26       $492.26
      918    BRANDOW       02200         $804.64       $579.35      $225.29
      918    EPERT         07781         $310.40       $310.40
```

```
919    DENNING     02765        $135.85      $135.85
920    MILLER      05914        $313.60      $313.60
```

Note that the records are now in order by department number. When you use SEQUENCE, you don't need to define any extra files or additional input/output commands in your program; CA-Easytrieve takes care of that for you.

The CONTROL Statement

The CONTROL statement defines a control break on a specified field (called the control field). It causes all quantitative fields (fields with decimal positions) to be totaled at the time of the control break and to be grand totaled at the end of the report.

Since we've sequenced our report by the DEPT field, we can also request a control break on the same field. This gives us totals of GROSS, NET-PAY, and DEDUCTIONS for each department. All we must do is add the CONTROL statement and the field name DEPT right after the SEQUENCE statement:

```
REPORT  PAY-RPT LINESIZE 80
  SEQUENCE DEPT
  CONTROL DEPT
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE  01 DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

Now at the end of each department (and end-of-report) we'll get our totals:

```
01/31/91              PERSONNEL REPORT EXAMPLE-1              PAGE      1

                      EMPLOYEE
   DEPT    EMPNAME     NUMBER      GROSS          NET-PAY      DEDUCTIONS

   901     WALTERS     11211         $424.00        $424.00
   901                               $424.00        $424.00

   903     WIMN        12267         $373.60        $373.60
   903                               $373.60        $373.60


   912     LOYAL       04225         $295.20        $295.20
   912                               $295.20        $295.20

   914     MANHART     11602         $344.80        $344.80
           VETTER      01895         $279.36        $279.36
           GRECO       07231       $1,004.00        $722.88      $281.12
           CROCI       08262         $376.00        $376.00
           RYAN        10961         $399.20        $399.20
   914                             $2,403.36      $2,122.24      $281.12

                                   $3,496.16      $3,215.04
```

The SUM Statement

Let's say you've decided you do not want totals for all three fields GROSS, NET-PAY, and DEDUCTIONS at each control break. All you really need is a total for GROSS so you can get an idea of what the salary expense is. You can override the CONTROL statement (which normally totals all quantitative fields) with the SUM statement.

The SUM statement specifies the quantitative fields you want totaled on a control break. When used, any fields not specified on the SUM statement are not totaled. Let's change our program so that it totals only the gross pay.

```
REPORT  PAY-RPT  LINESIZE  80
  SEQUENCE DEPT
  CONTROL  DEPT
  SUM      GROSS
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE  01 DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

Now, GROSS is the only field totaled:

```
01/31/91                PERSONNEL REPORT EXAMPLE-1              PAGE    1

                        EMPLOYEE
      DEPT   EMPNAME     NUMBER      GROSS        NET-PAY     DEDUCTIONS

      901    WALTERS     11211         $424.00      $424.00
      901                              $424.00

      903    WIMN        12267         $373.60      $373.60
      903                              $373.60


      912    LOYAL       04225         $295.20      $295.20
      912                              $295.20

      914    MANHART     11602         $344.80      $344.80
             VETTER      01895         $279.36      $279.36
             GRECO       07231       $1,004.00      $722.88      $281.12
             CROCI       08262         $376.00      $376.00
             RYAN        10961         $399.20      $399.20
      914                            $2,403.36
```

The TITLE Statement

The TITLE statement gives us the title of our report. We've been calling our report 'PERSONNEL REPORT EXAMPLE-1' all the way through the tutorial.

```
REPORT  PAY-RPT  LINESIZE  80
  SEQUENCE DEPT
  CONTROL  DEPT
  SUM      GROSS
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  LINE  01 DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

You can change this to any title you think appropriate. All you must do is include the word TITLE, followed by a title number, followed by your title in single quotes. The title number can be omitted when you have only one title; it simply defaults to 01. When you have more than one title (and TITLE statement) you need to number them in ascending order.

You've seen what this statement does on our example report, but we'll show you again in case you've forgotten. As shown above, the TITLE statement is responsible for the title shown on the following report:

```
01/31/91                PERSONNEL REPORT EXAMPLE-1           PAGE      1

                        EMPLOYEE
      DEPT   EMPNAME     NUMBER      GROSS        NET-PAY     DEDUCTIONS

      901    WALTERS     11211         $424.00      $424.00
      901                              $424.00

      903    WIMN        12267         $373.60      $373.60
      903                              $373.60
```

The system date and the page number are automatically printed on the same line. We'll show you how to override this later in Chapter 6, "Activity Section - Reporting."

The HEADING Statement

The HEADING statement like the HEADING parameter of the DEFINE statement (discussed in Lesson 3) prints user defined column headings for specified fields. (It overrides the HEADING parameter of the DEFINE statement if one already exists for the field you are describing.)

We can show you how this statement works by adding it to our program. Let's say we've decided that the field name, EMPNAME is not really a good column heading since what we really mean is EMPLOYEE NAME. Much like we did with the EMP# field, we can change our existing column heading.

All we do is type the word HEADING followed by the field name EMPNAME, followed by the new column heading:

```
REPORT  PAY-RPT  LINESIZE  80
  SEQUENCE DEPT
  CONTROL  DEPT
  SUM      GROSS
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  HEADING EMPNAME ('EMPLOYEE' 'NAME')
  LINE  01 DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

To be consistent with our other heading, EMPLOYEE NUMBER, we've described our new heading so that it stacks EMPLOYEE on top of NAME. This is done by putting single quotes around each word in the heading. The parentheses are required because the two words, each in single quotes, are treated the same as any other multiple parameters. Here's how it prints:

```
01/31/91                PERSONNEL REPORT EXAMPLE-1            PAGE     1

            EMPLOYEE    EMPLOYEE
DEPT          NAME       NUMBER      GROSS        NET-PAY     DEDUCTIONS

901         WALTERS      11211         $424.00      $424.00
901                                     $424.00

903         WIMN         12267         $373.60      $373.60
903                                     $373.60


912         LOYAL        04225         $295.20      $295.20
912                                     $295.20

914         MANHART      11602         $344.80      $344.80
            VETTER       01895         $279.36      $279.36
            GRECO        07231       $1,004.00      $722.88      $281.12
            CROCI        08262         $376.00      $376.00
            RYAN         10961         $399.20      $399.20
914                                   $2,403.36
```

The LINE Statement

The last report definition statement is one you've seen, along with TITLE, since the beginning of this chapter. We discussed it briefly in Lesson 3.

The LINE statement defines the contents of a printed line (detail line) in your report. In our example program, it defines which fields we want printed on a line and the order we want them printed in:

```
REPORT  PAY-RPT  LINESIZE  80
  SEQUENCE DEPT
  CONTROL  DEPT
  SUM      GROSS
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  HEADING EMPNAME ('EMPLOYEE' 'NAME')
  LINE DEPT EMPNAME EMP# GROSS NET-PAY DEDUCTIONS
```

The LINE statement is the only report definition statement you are required to include in your report declaration. Without it, CA-Easytrieve has no idea what detail information you want printed on your report or in what order you want it printed.

Reviewing Report Declarations

With the completion of this lesson you have now seen what roles the REPORT, SEQUENCE, CONTROL, SUM, TITLE, HEADING, and LINE statements play in the creation of a CA-Easytrieve Report. In fact, you now have all the information you need to write your own standard reports by customizing what you've learned so far in this tutorial. You now know that:

- REPORT designates the beginning of a report declaration and can specify the type of report and report characteristics.

- SEQUENCE puts your report in alphabetical or numerical order, based on the contents of a field or fields.
- CONTROL causes a control break, based on the contents of a field. It causes the printing of control totals and grand totals for all quantitative fields.
- SUM overrides control totals and causes totals only for specified fields.
- TITLE causes the printing of major report titles.
- HEADING causes the printing of customized column headings.
- LINE tells CA-Easytrieve what fields to put on detail lines and in what order.

For More Information

To add more detail to your understanding of CA-Easytrieve report declarations, turn to Chapter 6, "Activity Section - Reporting."

The next three lessons in this tutorial introduce you to CA-Easytrieve SCREEN activities. To continue, proceed to the next lesson.

Lesson 5

CA-Easytrieve provides all the facilities necessary to display and receive information from an online terminal. As with other features, CA-Easytrieve non-procedural nature provides relief from having to deal with many of the complexities of online programming.

Note: In the following discussion of screen processing, the term *you* refers to the programmer of the application. *Terminal user* refers to the person executing the application program.

Basic Structure

A SCREEN activity defines the method you use to describe and process an online screen display. The CA-Easytrieve screen processing facility is basically declarative; you need only 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:

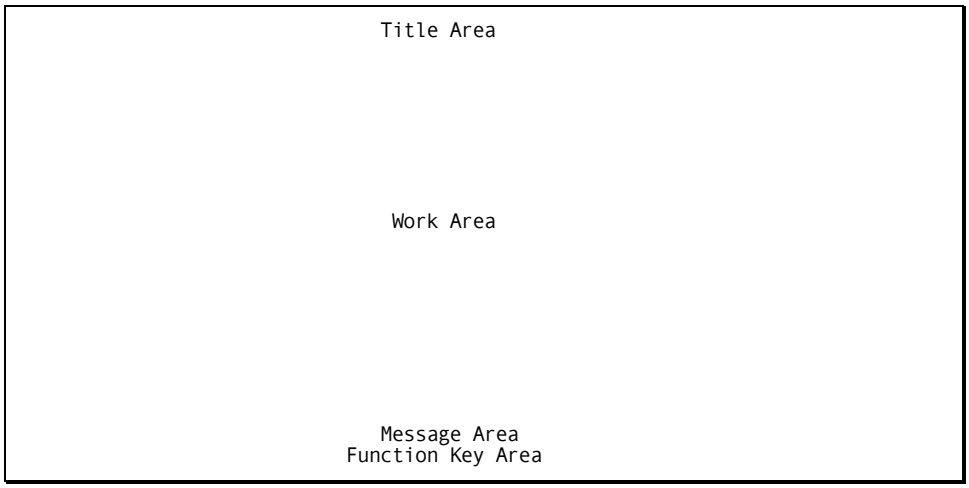
- The screen declaration statements that define the contents of the screen.
- The optional screen procedures that provide the opportunity to code procedural logic to perform file I/O or complex editing.

The following illustrates the basic structure of a CA-Easytrieve program with screen processing. You can define one or more screens for each program.

```
FILE
   (library section)

SCREEN NAME SCREEN1
  Screen Declaration
  Screen Procedures

SCREEN NAME SCREEN2
  Screen Declaration
  Screen Procedures
```

Screen Format

CA-Easytrieve screen format is illustrated below.



Title Area

This optional area consists of screen rows designated as titles by TITLE statements in the screen declaration. Titles normally identify the screen to the user and are located at the top of the screen and automatically centered. 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.

Message Area

The message area is used to display system and programmer-issued messages to the terminal user. The message area default location is the bottom of the screen and just above the function key display area.

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. If used, this area is always located at the bottom of the screen. The display is determined from the KEY statements in the screen declaration.

Sample Screen Program

For the purposes of this tutorial, you start with a very simple screen program. The program shows the types of CA-Easytrieve statements used to produce an online screen application.

```
DEFINE FIELD1 W 8 N 2                    } Library
DEFINE FIELD2 W 8 N 2                    }
DEFINE RESULT W 8 N 2                    }

SCREEN NAME ADDING-MACHINE               } SCREEN Statement

  TITLE 'ADDING MACHINE'                 }
  ROW 5 'FIELD 1. . . .' FIELD1          }
  ROW 7 'FIELD 2. . . .' FIELD2          } Screen Declaration
  ROW 9 'RESULT . . . .' RESULT          }
  KEY ENTER                              }
  KEY F3 EXIT NAME 'Exit'                }

  AFTER-SCREEN. PROC                     }
    RESULT = FIELD1 + FIELD2             } Screen Procedure
  END-PROC                               }
```

This is all the code you need to display a formatted screen that allows you to enter two numbers into Field 1 and Field 2. Press Enter and display the results of the addition of the two numbers.

The Screen Your Program Creates

Note: The title of your screen is automatically centered at the top of the screen and the function keys are shown on the bottom line of the screen.

```
                              ADDING MACHINE


FIELD 1. . . .          .00

FIELD 2. . . .          .00

RESULT . . . .          .00
```

```
F3=Exit
```

Throughout this lesson and Lessons 6 and 7, we expand on this simple sample program to demonstrate the ease of creating your own screen applications with CA-Easytrieve.

The first three lines of the sample program are DEFINE statements. You recall from Lesson 1 that all fields used in any CA-Easytrieve program must be defined to the system. A SCREEN activity is no different in that requirement.

This particular program does not need a FILE statement because the numbers you enter, and the results, are never permanently stored in a file. Most screen applications do require some kind of FILE statement.

The SCREEN statement is the start of the SCREEN activity. This statement signals to CA-Easytrieve that it is creating a screen to be displayed on a terminal.

The TITLE, ROW, and KEY statements declare what screen items appear on the screen. A screen item is any literal or field you want to display on the screen. The KEY statement also defines valid terminal keys for the screen.

The AFTER-SCREEN procedure identifies the processing that occurs after the Enter key is pressed. The lessons in this tutorial deal primarily with the screen declaration section of the SCREEN activity. For more information on screen procedures refer to Chapter 7, "Activity Section - Screens."

The following discussion details the workings of the TITLE, ROW, KEY, and AFTER-SCREEN in the program.

```
DEFINE FIELD1 W 8 N 2
DEFINE FIELD2 W 8 N 2
DEFINE RESULT W 8 N 2
SCREEN NAME ADDING-MACHINE
  TITLE 'ADDING MACHINE'
  ROW 5 'FIELD 1. . . .' FIELD1
  ROW 7 'FIELD 2. . . .' FIELD2
  ROW 9 'RESULT . . . .' RESULT
  KEY ENTER
  KEY F3 EXIT NAME 'Exit'
  AFTER-SCREEN. PROC
    RESULT = FIELD1 + FIELD2
  END-PROC
```

TITLE Statement

The title of the screen you are creating is ADDING MACHINE.

This title is created with the CA-Easytrieve TITLE statement shown below.

```
TITLE 'ADDING MACHINE'
```

Note: The words ADDING MACHINE are enclosed in single quotes because you are using a literal and CA-Easytrieve requires that literals be enclosed in single quotes. CA-Easytrieve displays literals on the screen exactly as you typed them.

The fields and literals on a TITLE statement are automatically protected from user changes. CA-Easytrieve also automatically centers the title on the top line of the terminal screen unless you specify a different location.

ROW Statement

The ROW statement defines the items that appear on the screen after the title.

```
ROW 5 'FIELD 1. . . .' FIELD1
ROW 7 'FIELD 2. . . .' FIELD2
ROW 9 'RESULT . . . .' RESULT
```

The numbers appearing after the ROW keyword identify in which row (or line) of the terminal screen the item is displayed. If you do not specify a row number, CA-Easytrieve places the item on the first line following the title and continues to place items in subsequent ROW statements one line down.

You can code a literal on the ROW statement to provide information on the screen to guide the user as to what the contents of the following field should be, or to provide directions on what to do on the screen. Like literals on TITLE statements, these literals cannot be altered by your application users.

FIELD1 and FIELD2 in the library section of your program are working storage fields that temporarily store data the application user enters in the field. When you code a field name on a ROW statement, CA-Easytrieve leaves a space on the screen the same length as the length specified on the DEFINE statement for that field (with a MASK, if applicable). Data can then be entered or displayed in this space.

KEY Statement

The KEY statement defines which programmable keys you want to be valid in your screen applications. Programmable keys are the keys on your keyboard that can be programmed to perform a function, such as F1, to bring up a help screen or Enter to begin processing. For example:

```
KEY ENTER
KEY F3 EXIT NAME 'Exit'
```

Only two keys are valid on the Sample Program screen, F3 and Enter. Because of the NAME parameter, only F3=Exit is displayed on the screen.

The NAME parameter of the KEY statement tells CA-Easytrieve to display the key name with an equal (=) sign and a literal after it in the function key area of the screen. The following statement shows the change you must make to the first KEY statement to have CA-Easytrieve add the Enter key to the screen display:

```
KEY ENTER NAME 'Compute'
```

The literal 'Compute' is displayed after the equal sign to inform application users what happens when they press the Enter key. In this case, pressing Enter invokes the AFTER-SCREEN procedure that calculates the addition of the two fields. The new screen is shown below.

```
                              ADDING MACHINE


 FIELD 1. . . .          .00
 FIELD 2. . . .          .00
 RESULT . . . .          .00









 ENTER=Compute  F3=Exit
```

Note: ENTER=Compute is now displayed in the function key area on the terminal screen. CA-Easytrieve automatically displays the keys in the order they were coded on the KEY statements. Since the first KEY statement was for the **Enter** key, it appears as the first key displayed in the function key area.

The second KEY statement:

```
KEY F3 EXIT NAME 'Exit'
```

shows one of the automatic functions available in CA-Easytrieve. When you use the EXIT parameter of the KEY statement for a particular key, you tell CA-Easytrieve to terminate the screen activity whenever the designated key is pressed. The data in the screen fields are edited and moved into the program fields.

Review of Screen Activities

In this lesson of the tutorial, we have discussed the TITLE, ROW, and KEY statements to process an online screen display. You know that:

- TITLE causes the display of a descriptive screen title.

- ROW determines where the items specified appear in the screen work area.
- KEY designates the function keys to be programmed to perform a specific action in your screen application.

For More Information

For more information about SCREEN activities, turn to Chapter 7, "Activity Section - Screens."

To continue the tutorial, proceed to the next lesson.

Lesson 6

Your screen program is fully functional, but now, let's make it easier to use by adding parameters to the ROW statement to:

- Change the color of the RESULT field.
- Apply an edit mask to the data fields.

Changing Attributes (ATTR Parameter)

CA-Easytrieve assigns default attributes to all the screen items. An attribute refers to the way an item is displayed on the screen. Some of the attributes are color, underlining, and intensity. You can override these default attributes with the ATTR parameter of the TITLE and ROW statements.

To make the RESULT field on your screen more obvious when the computation is completed, we assign an attribute of WHITE to the ROW 9 statement.

```
DEFINE FIELD1 W 8 N 2
DEFINE FIELD2 W 8 N 2
DEFINE RESULT W 8 N 2
SCREEN NAME ADDING-MACHINE
  TITLE 'ADDING MACHINE'
  ROW 5 'FIELD 1. . . .' FIELD1
  ROW 7 'FIELD 2. . . .' FIELD2
  ROW 9 'RESULT . . . .' RESULT ATTR WHITE
  KEY ENTER NAME 'Compute'
  KEY F3 EXIT NAME 'Exit'
  AFTER-SCREEN. PROC
    RESULT = FIELD1 + FIELD2
  END-PROC
```

All other fields and literals are displayed in the default colors specified in your Site Options for screens. See your system administrator for more information on the screen attribute defaults at your site.

Edit Masks

An edit mask is a pattern of characters specifing how numeric data is displayed on the terminal screen. (Alphanumeric fields cannot be edited.)

To give an example, let's add an edit mask to the three fields in our example program so that they display with a leading digit before the decimal point.

```
DEFINE FIELD1 W 8 N 2
DEFINE FIELD2 W 8 N 2
DEFINE RESULT W 8 N 2
SCREEN NAME ADDING-MACHINE
  TITLE 'ADDING MACHINE'
  ROW 5 'FIELD 1. . . .' FIELD1            MASK (A 'ZZZ,ZZ9.99')
  ROW 7 'FIELD 2. . . .' FIELD2            MASK A
  ROW 9 'RESULT . . . .' RESULT ATTR WHITE MASK A
  KEY ENTER NAME 'Compute'
  KEY F3 EXIT NAME 'Exit'
  AFTER-SCREEN. PROC
    RESULT = FIELD1 + FIELD2
  END-PROC
```

The new keyword, MASK, is a parameter of the ROW statement and designates that an edit mask is to follow. In the above example, the actual edit mask consists of the characters 'ZZZ,ZZ9.99'. Masks are always enclosed in single quotes.

The effect of adding the masks is as follows:

```
                              ADDING MACHINE


FIELD 1. . . .       0.00

FIELD 2. . . .       0.00

RESULT . . . .       0.00









ENTER=Compute  F3=Exit
```

Note: The leading zero is before the decimal point.

The following explanations and rules apply to the edit masks in our example:

- Each digit in a field must be designated in the edit mask. Since all three fields are eight digits in length with two decimal places, we must designate eight digits in the mask. This is done with ZZZ,ZZ9.99.

- The Zs in the edit mask indicate that leading zeros in the field should not be displayed on the screen. For example:

Mask	Field Value	Resulting Output
'ZZZ,ZZ9.99'	1234567	12,345.67
	0123456	1,234.56
	0012345	123.45
	0001234	12.34
	0000123	1.23
	0000012	0.12

- The digit 9 indicates that any value occurring in that position is printed as a digit. In the above example, all values in the *ones* column or to the right of the decimal are printed as digits, even zeros. Since CA-Easytrieve initialized these fields to zero, a zero is displayed to the left of the decimal point and two zeros are displayed to the right of the decimal point. All other positions in the field are blank because we specified that leading zeros be suppressed with the Z character.
- Commas and decimal points are displayed just as indicated in the mask.
- When the same mask is to be used on more than one field, you can avoid coding the mask more than once by naming it, and then specifying only the name on subsequent fields. Names can be any letter from A through Y. As shown in the program code, all fields have the same mask, so the mask on the first ROW statement was given the name A. Then, the name A was coded on the remaining ROW statements to use the same mask.

Review of Changing Attributes

In this lesson of the tutorial, we discussed improving the readability of your screen by adding the ATTR and MASK parameters to the ROW statement. You learned that:

- ATTR enables you to change the color of the field.
- MASK enables you to specify how data is to be displayed on the screen.

For More Information

For more information on the ROW statement refer to Chapter 7, "Activity Section - Screens."

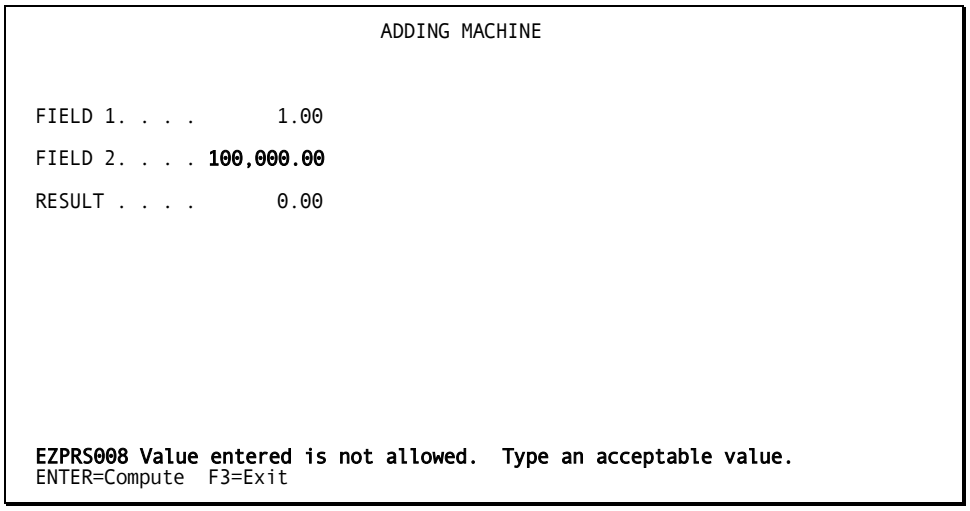
Lesson 7

The field defined to hold the numbers for the calculation and the results of the addition are only eight bytes in length. This length limits the size of the result that can be calculated to a maximum of 99,999.99. Rather than have the result truncate when entered numbers exceed this limit, CA-Easytrieve lets you specify a range of values on the ROW statement.

```
DEFINE FIELD1 W 8 N 2
DEFINE FIELD2 W 8 N 2
DEFINE RESULT W 8 N 2
SCREEN NAME ADDING-MACHINE
  TITLE 'ADDING MACHINE'
  ROW 5 'FIELD 1. . . .' FIELD1            MASK (A 'ZZZ,ZZ9.99') +
    VALUE (0 THRU 99999.99)
  ROW 7 'FIELD 2. . . .' FIELD2            MASK A +
    VALUE (0 THRU 99999.99)
  ROW 9 'RESULT . . . .' RESULT ATTR WHITE MASK A
  KEY ENTER NAME 'Compute'
  KEY F3 EXIT NAME 'Exit'
  AFTER-SCREEN. PROC
    RESULT  FIELD1 + FIELD2
  END-PROC
```

You can use the VALUE parameter of the ROW statement to specify a single value, a series of values, or a range of values. In this case, we have specified a range of values, 0 through 99999.99. Now, the user is prevented from specifying any number outside this range.

The following example shows how the screen looks when a user exceeds the range of values specified on the ROW statement.



When the VALUE parameter is used, you have the option of permitting CA-Easytrieve to generate an error message, or you can code your own message. Here, we are showing the message CA-Easytrieve provides. Since this message only conveys that the range has been exceeded, let's code our own message that identifies the error and gives the range required. We do this by adding an ERROR parameter to the ROW statement.

Creating Error Messages

Using the ERROR parameter enables you to specify one or more fields or alphanumeric literals to be used as an error message to be issued by CA-Easytrieve, in case of an automatically detected error condition. These messages replace the more generic messages issued by CA-Easytrieve if ERROR is not coded.

```
DEFINE FIELD1 W 8 N 2
DEFINE FIELD2 W 8 N 2
DEFINE RESULT W 8 N 2
SCREEN NAME ADDING-MACHINE
  TITLE 'ADDING MACHINE'
  ROW 5 'FIELD 1. . . .' FIELD1           MASK (A 'ZZZ,ZZ9.99') +
    VALUE (0 THRU 99999.99) ERROR 'MUST BE LESS THAN OR EQUAL +
                                  TO 99,999.99'
  ROW 7 'FIELD 2. . . .' FIELD2            MASK A +
    VALUE (0 THRU 99999.99) ERROR 'MUST BE LESS THAN OR EQUAL +
                                  TO 99,999.99'
  ROW 9 'RESULT . . . .' RESULT ATTR WHITE MASK A
  KEY ENTER NAME 'Compute'
  KEY F3 EXIT NAME 'Exit'
  AFTER-SCREEN. PROC
    RESULT  FIELD1 + FIELD2
  END-PROC
```

Our example uses the literal:

```
ERROR 'MUST BE LESS THAN OR EQUAL TO 99,999.99'
```

to inform the user of the limits of the program fields when an incorrect value is entered in either field.

The following example shows how the final screen looks when you code your own error message.

```
                          ADDING MACHINE


FIELD 1. . . .       1.23

FIELD 2. . . . 100,000.00

RESULT . . . .       0.00
```

```
MUST BE LESS THAN OR EQUAL TO 99,999.99
ENTER=Compute  F3=Exit
```

As you can see, it is very simple to create screen applications using CA-Easytrieve.

Reviewing Error Messages

In this lesson of the tutorial, we have discussed how to use VALUE and ERROR parameters of the ROW statement. You have learned that:

- VALUE lets you specify a range of values and prevents the user from specifying any number outside this range.
- ERROR enables you to customize your own error message.

For More Information

For more information on the use of the VALUE and ERROR parameters of the ROW statement, refer to Chapter 7, "Activity Section - Screens."

Lesson 8

CA-Easytrieve provides all the facilities necessary for producing bit-mapped presentation graphs with a non-procedural technique very similar to reporting. The styles of graphs available include pie charts, bar charts, line graphs, and scatter diagrams. The graph facility controls the format of the graph making assumptions based on best-fit.

Note: Graph processing is available only when using CA-Easytrieve/Workstation.

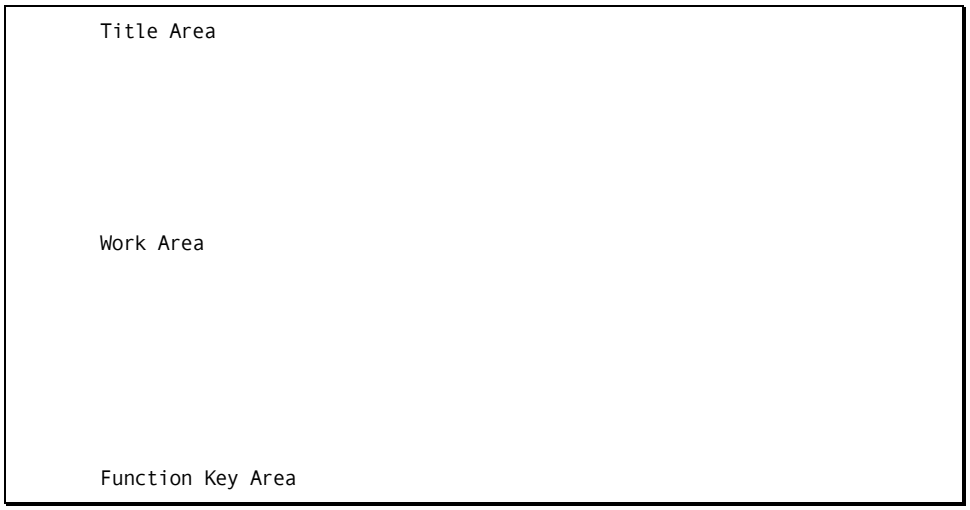
Basic Structure

The CA-Easytrieve graph facility is fully declarative; you need only define the style and content of the graph and CA-Easytrieve creates the instructions to produce it. The following illustrates the basic structure of a CA-Easytrieve JOB with graph processing. You can define one or more graphs for each JOB activity.

```
            FILE
              (library)
INPUT
DATA  ------>JOB
              (job activity)
              DRAW GRAPH1
              DRAW GRAPH2

            GRAPH GRAPH1

            GRAPH GRAPH2
```

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.

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.

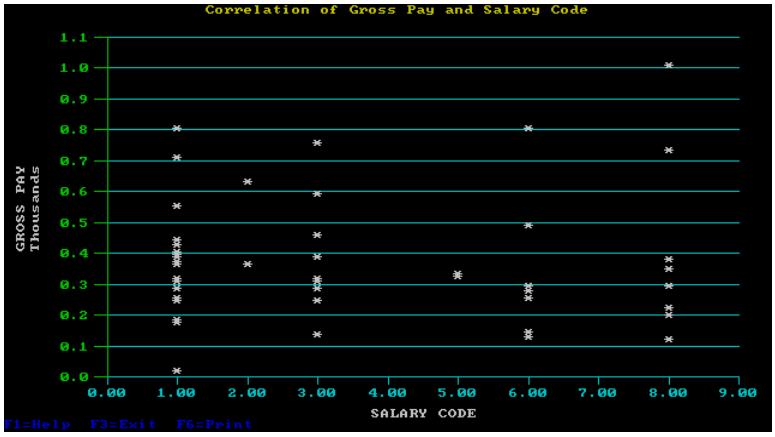
Sample Graph Program

The following program produces a scatter diagram showing the correlation between gross pay and salary code.

```
FILE PERSNL FB(150 1800)
  PAY-GROSS    94  4  P  2  HEADING ('GROSS' 'PAY')
  SALARY-CODE 134  2  N  2  HEADING ('SALARY' 'CODE')

JOB INPUT PERSNL NAME DISPLAY-GRAPH
  DRAW GROSS-VS-SALARY-CODE                          } DRAW Statement

GRAPH GROSS-VS-SALARY-CODE STYLE 'SCATTER'           } GRAPH Statement

  TITLE 'Correlation of Gross Pay and Salary Code'   }
  VALUE SALARY-CODE PAY-GROSS                        } GRAPH Definition Statements
```

The Graph Your Program Creates

Note: The title of your graph is automatically centered at the top of the graph and function keys are shown at the bottom of the graph.



GRAPH Statement

You define a graph in CA-Easytrieve by coding a GRAPH statement followed by a series of graph definition statements (for example TITLE and VALUE). You must code the GRAPH statement first in a GRAPH declaration. The GRAPH statement establishes the style and characteristics of the graph. In this example, we specified that we wanted to create a scatter diagram. Other types of graphs you can create are:

- Pie charts
- Bar charts (vertical and horizontal)
- Line graphs

If you do not specify the type of graph you want to create, CA-Easytrieve automatically creates a pie chart.

Graph Definition Statements

A set of graph definition statements defines every CA-Easytrieve graph. The statements define the format and data content. We are using the following graph definition statements in our sample program:

TITLE Statement

The title of the graph you are creating is Correlation of Gross Pay and Salary Code. This title is created with the CA-Easytrieve TITLE statement shown below.

```
TITLE 'Correlation of Gross Pay and Salary Code'
```

Note: The title is enclosed in single quotes on the TITLE statement. This is because you are using a literal and CA-Easytrieve requires that literals be enclosed in single quotes. CA-Easytrieve displays the literal on the screen exactly as you have typed it.

The fields and literals on a TITLE statement are automatically protected from user changes. CA-Easytrieve also automatically centers the title on the top line of the graph unless you specify a different location.

VALUE Statement

The VALUE statement specifies the fields to be used to draw the graph.

First, you must specify the field or literal to be used for drawing the horizontal (x-value) axis of the graph. Because we are drawing a scatter graph, the x-value must be a numeric field or literal. This value determines the position of the data point on the x-axis. We are using the SALARY-CODE field for the x-axis of our graph.

Next, you specify the field or literal to be used for drawing the vertical (y-value) axis of the graph. Each field or literal must be numeric. This value determines the position of the data point on the y-axis. We are using the PAY-GROSS field for the y-axis of our graph.

The x- and y-axis of the scatter graph are created with the VALUE statement shown below:

```
VALUE SALARY-CODE PAY-GROSS
```

DRAW Statement Processing

The DRAW statement produces graphic output by invoking the graph declaration. CA-Easytrieve extracts the data required for the requested graph. At the end of the job, CA-Easytrieve formats the data in the specified manner, and sends it to the terminal for display and, optionally, printing.

The name of the graph, as specified on the GRAPH statement, is specified on the DRAW statement. The name of the graph you are creating is GROSS-VS-SALARY-CODE. This graph is created with the CA-Easytrieve DRAW statement, shown below:

```
DRAW GROSS-VS-SALARY-CODE
```

You do not have to specify the name of the graph on the DRAW statement. If not specified, CA-Easytrieve assumes that the name of the graph you want to draw is the first graph in the JOB activity.

Review of Graph Declarations

In this lesson of the tutorial, we have discussed the GRAPH, TITLE, VALUE, and DRAW statements to draw a scatter diagram. You know that:

- GRAPH specifies the name and style of the graph.
- TITLE causes the display of a descriptive graph title.
- VALUE defines the fields to be used to draw the graph.
- DRAW produces the graphic output by drawing the values on the graph.

For More Information

For more information about graph activities, turn to Chapter 8, "Activity Section - Graphs."

Summing Things Up

In this tutorial, you have followed the development of two moderately complex CA-Easytrieve programs. We started with a very simple report and built on it until we used all of the basic report writing features. Then, we walked through a simple screen program and a simple graph program.

You should now have a good understanding of CA-Easytrieve program structure as well as how to use most of the basic tools CA-Easytrieve has to offer.

There are many parameters of various statements we chose not to illustrate in the tutorial. Our goal was to get you through the basic programming process. These other parameters and statements are covered in other portions of this guide or in the *CA-Easytrieve Language Reference Guide.*

For More Information

Read any first level readings in Chapter 3 through Chapter 9 that you have skipped. Or, start the second reading in Chapter 3, "Library Section - Describing and Defining Data."

Chapter

3

Library Section - Describing and Defining Data

Introduction

Describing and defining data is essential to creating CA-Easytrieve programs that use input or output files. CA-Easytrieve must know how data is stored before processing can occur. Data definition is accomplished through the FILE and DEFINE statements.

Before going into discussions of specific CA-Easytrieve statements, this chapter illustrates some general rules of syntax. In this chapter, you'll find:

⇒ **Reading 1**

- CA-Easytrieve general syntax rules
- Describing files with the FILE and DEFINE statements
- Editing fields and adding headings
- Defining working storage fields (type W)

⇒ **Reading 2**

- Defining working storage fields (type S)
- Initializing working storage fields with the VALUE clause
- Redefining fields
- Defining fields with a relative start location
- Redefining fields using relative start locations

⇒ **Reading 3**

- Using advanced FILE statement parameters including VFM
- Copying field definitions with the COPY statement

⇒ **Reading 1 of Chapter 3 starts here.**

CA-Easytrieve Syntax Rules

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

Statement Area

All CA-Easytrieve source statements are records of 80 characters each. The default statement area is in columns 1 through 72. This means you can place your CA-Easytrieve code anywhere within columns 1 through 72. You'll probably want to indent or line up certain statements for readability, but it's not required.

Multiple Statements

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

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

Comments

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

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

Continuations

The last non-blank character of a statement terminates the statement unless that character is a minus (-) or a plus sign (+).

- The - indicates that the statement continues at the start of the next statement area.
- The + indicates that the statement continues with the first non-blank character in the next statement area.

The difference between - and + is important only when continuing a line in the middle of a word. Continuation of a line between words is the same for both. The following continued statements produce identical results:

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


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

Words and Delimiters

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

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

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

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

Keywords

Keywords are words having specific meaning to CA-Easytrieve. Some keywords are reserved words. You can use non-reserved keywords in the appropriate context as field names, whereas reserved words cannot be used as field names. See Appendix B in the *CA-Easytrieve Language Reference Guide* for a list of all reserved keywords.

Multiple Parameters

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

```
MASK (A  BWZ  '$$,$$9.99')
```

Field Names

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

- Alphabetic characters, A through Z, lower and upper case
- Decimal digits 0 through 9
- All special characters, except delimiters.

The first character of a field name must be an alphabetic character, a decimal digit, or a national character (#, @, $). In addition, a field name must contain at least one alphabetic or special character to distinguish the field name from a number. All working storage field names must be unique, as well as all field names within a single file. If you use the same field name in more than one file, or in a file and in working storage, you must qualify the field name with the file name or the word WORK. A qualified field name consists of the qualifying word followed by a colon and the field name. You can use any number of spaces, or no spaces, to separate the colon from either the qualifying word or the field name.

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

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

Labels

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

Identifiers

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

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

Arithmetic Operators

CA-Easytrieve arithmetic expressions (see Chapter 4, "Activity Section - Processing and Logic") use the following arithmetic operators:

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

The arithmetic operator must lie between two spaces.

Alphanumeric Literals

Alphanumeric literals are words meant to be taken literally. They are enclosed within single quotes, and can be 254 characters long. An alphanumeric literal can contain 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'
```

Numeric Literals

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

```
123
```

```
+123
-123.4321
```

Hexadecimal Literals

Hexadecimal literals are words used to code values that contain characters not available on standard data entry keyboards. Prefix a hexadecimal literal with the letter X and a single quote (X'), 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 to 9 and the letters A to F. The following hexadecimal literal defines two bytes of binary zeros:

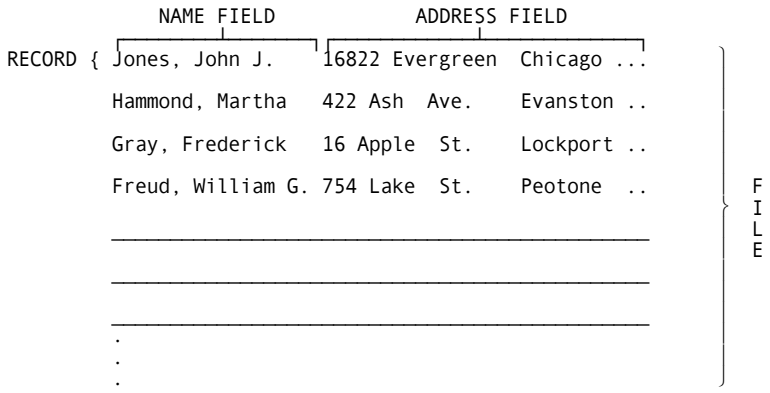
```
X'0000'
```

Describing Files and Fields

All of the files and their associated fields (as well as working storage fields) referenced in your CA-Easytrieve program must be described before they are referenced.

Defining Data

You normally define data fields in the section of your program called the library. The library defines the data in terms of fields, records, and files. A typical file layout is shown below:



Defining File Attributes

The FILE statement is used to describe a file or a database.

Defining Field Data

Fields are defined in the library following the FILE statement, or later in the job activity, by using the DEFINE statement. Two categories of data can be defined:

- File data (fields defined within a record).
- Working storage data (fields defined in working storage).

FILE Statement

The FILE statement describes the files you are using as input to your program and any files your program creates (output) other than reports. You code the FILE statement(s) at the beginning of the library section.

Syntax

```
FILE file-name [file attributes]
```

FILE is the keyword which designates that a file name and description are to follow. *File-name* and *file attributes* describe the file you are using and are normally supplied by your data processing department.

Parameters

file-name This is a one-to 128-character name used to define your file to CA-Easytrieve. All statements that operate on the file refer to this name. *File-name* is also normally used on your JCL, CLIST, or EXEC statements to reference the file. Every FILE statement must have a *file-name* immediately following the FILE keyword and it must be unique within your program.

file attributes The CA-Easytrieve FILE statement has a host of parameters which describe *file attributes*. *File attributes* are as varied as the methods and environments available for storing data. Most of them are beyond the scope of this guide. In general, they include parameters for describing file type, storage device type, and record format. They are all optional and depend on the particular environment in which you are operating. See the *CA-Easytrieve Language Reference Guide* or *Pocket Language Reference Guide* for complete FILE statement syntax.

DEFINE Statement

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

- Four parameters are always required: *field-name*, *start-location*, *field-length*, and *data-type*.
- Additional parameters include the number of decimal positions for quantitative fields, HEADING, and MASK.

Syntax

```
DEFINE field-name   start-location   field-length    +
data-type  [decimal-positions]  [HEADING 'heading-literal']  +
[MASK     {[mask-identifier] [BWZ] ['mask-literal']}]
```

Parameters

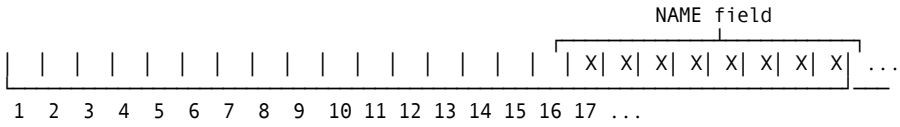
field-name You create your own *field-names* or use already existing field names (provided to you in a record layout).

- *Field-names* must be unique within a file.
- The name cannot be all numeric characters.
- The name can be one to 128 alphanumeric characters in length.
- The name must begin with A-Z, 0-9, or a national character (#, @, $).
- Special characters, such as dollar sign and hyphen, can be used, but not delimiters.

start-location The start location is the beginning location of a field in a record relative to the first position (position 1) of the record. Start location can be explicitly defined based on its distance from position 1 of the record:

```
NAME      17   starts in position 17
ADDRESS   37   starts in position 37
PAY-NET   90   starts in position 90
```

Here is an example of where the NAME field would appear in the record just described:



field-length You specify the length of a field in bytes (characters and/or spaces). The length of the NAME field, in the above example, is eight characters.

```
NAME    17    8
```

data-type You describe the type of data a field contains by coding the letter abbreviation for that type after the *field-length*. There are seven *data-types*.

Type	Maximum Field Length
A Alphanumeric	32,767
N Numeric	18
P Packed	10
U Unsigned Packed	9
B Binary	4
I Integer (Workstation only)	4
F Fixed Point ASCII (Workstation only)	19

The data type of the NAME field (which probably contains only alphabetic characters) is A for alphanumeric.

```
NAME    17    8    A
```

decimal-positions This is an option that specifies the desired number of decimal positions for a field name. By specifying decimal positions in your field description, you:

- Identify a field to CA-Easytrieve as being quantitative. (A field that contains a quantity, as opposed to a numeric identifier or code.)
- Identify the field to be automatically totaled when specified in a CONTROL report.
- Allow for proper placement of commas and decimals with leading (high-order) zeros suppressed when the field is printed.

Five types of data can have decimal positions:

N (Numeric)
P (Packed)
B (Binary)
U (Unsigned Packed)
F (Fixed Point ASCII - Workstation only)

Specify the *decimal-positions* by coding an integer (0 through 18) after the *data-type*. For example:

```
AMOUNT   40   5   N 2
```

is a five-byte numeric field with two decimal positions.

HEADING You use the HEADING parameter to specify an alternative column heading for a field. (The default column heading is the *field-name*.) The column heading you specify is automatically used on report output unless overridden by a HEADING statement in the activity section.

Place the alternate column heading within single quotes. For example:

```
CL-NAME    5   20   A   HEADING 'CLIENT NAME'
```

produces the column heading:

```
CLIENT NAME
```

To "stack" a column heading, place each word in single quotes. You now must enclose the words in parentheses. For example:

```
CL-NAME   S   20   A   HEADING ('CLIENT' 'NAME')
```

produces the column heading:

```
CLIENT
NAME
```

MASK The MASK parameter is used to create a customized edit mask. An edit mask is an optional pattern of characters specifying how numeric data is to be printed. (Alphanumeric fields cannot be edited.) An edit mask is created using combinations of the following characters:

Character	Meaning
9	Formats digits
Z	Suppresses Leading zeros
*	Replaces leading zeros with an asterisk
-	Prints a minus sign prior to the first non-zero digit of a negative number
$	Prints a currency symbol prior to the first non-zero digit

Each digit in the field must be designated by a character in the mask. For example:

Edit Mask	Field Value	Result
`$$,$$9`	`01234`	`$1,234`
`$$,$$9`	`93142`	`$93,142`

Commas can be included in the edit mask for clarity. They are printed in whatever location you indicate in the mask but are suppressed if the field value does not exceed the number of places to the right of the comma.

Defining Edit Masks

Some standard edit masks you might use in your programs are shown here:

Edit Mask	Used For
'(999)999-9999'	Telephone Number
'999-99-9999	Social Security number
'Z9/99/99'	Date
'$$,$$$,$$9.99 CREDIT'	Money (with floating $)
'*,***,***,999.99-'	Protected Check Amount
'-,---,--9.99'	Negative Number

Syntax

```
[MASK   {[mask-identifier]  [BWZ]    ['mask-literal']}]
```

- MASK is the CA-Easytrieve keyword that indicates an edit mask is to follow.
- The *mask-identifier* is used to name the edit mask that follows it. If you name a mask, you can reuse it on other field definitions just by specifying the name. A name can be any single letter from A through Y. This means that once you have defined a mask, you don't have to define it again to use it again.
- BWZ (blank when zero) specifies that a field should not be printed if the entire field contains zeros. Just code the letters BWZ whenever you want to suppress an all zero field. BWZ is not carried over to other fields when using a *mask-identifier*.
- The *mask-literal* is the actual format of the mask. It must be enclosed in single quotes and include one edit character for each digit in the field being described.

Examples of Edit Masks

Given a numeric field with the contents 012345678, the following masks produce the results shown:

Mask	Result
'999-99-9999'	012-34-5678
'Z99,999,999'	12,345,678
'ZZZ,ZZZ,999'	12,345,678
'$$$,$$$,999'	$12,345,678
'***,***,999'	*12,345,678

Masking Negative Values

Fields that have the potential for containing a negative value can be masked in such a way that an indicator of their negativity is displayed when printed. An indicator of negativity, such as minus sign (-), or the letters CR (for credit), or any other chosen indicator prints only when the field contains a negative value. To do this, mask the field as you would normally, making sure all digits are accounted for, then add the indicator to the right end of the mask.

Given a numeric field with the contents -012345678, the following masks produce the results shown:

Mask	Result
`'$$$,$$$,999 CREDIT'`	`$12,345,678 CREDIT`
`'$$$,$$$,999-'`	`$12,345,678-`
`'Z99,999,999-'`	`12,345,678-`

The indicators, shown above ("CREDIT" and "-"), only print when the field contains a negative value.

Default Edit Masks

Quantitative fields (fields defined with positions to the right of a decimal point) have system default edit masks which account for the automatic printing of commas and decimal points in printed totals.

Numeric fields that have no decimal positions defined are printed without commas or decimal points and are not automatically totalled on control reports.

Assuming a field named PAY has a value of 1000, the following table gives the corresponding default edit masks and results for some possible field definitions:

Field Definition	Default Mask	Result
`PAY 10 5 N 0`	`'ZZ,ZZZ-'`	`1,000`
`PAY 10 5 N 2`	`'ZZZ.99-'`	`10.00`
`PAY 10 5 N`	`'99999'`	`01000`

Note: The number of decimal positions can be zero (0).

Defining Working Storage

Working storage gives you a method for setting aside a temporary area of storage in the computer memory; a place to keep the results of calculations or other information that is created during the running of a CA-Easytrieve program.

Define working storage by specifying W as the start location. For example:

```
WORK-DEDUCT    W   4   N 2
```

defines a numeric working storage field four characters long with two decimal positions. This field could be defined in the library section or in an activity prior to being referenced.

DEFINE within an Activity

You usually specify file fields and working storage fields in your CA-Easytrieve library section, but you can also define them within an activity.

Compare the two examples below. The first shows DEFINE statements in the library section, the second shows DEFINE statements in an activity section. Remember, the DEFINE keyword is optional when defining fields in the library section.

The following example shows fields defined in the library section of a program. (The keyword DEFINE is shown but is optional.) There are no fields defined in the activity section.

```
             ⎧  FILE PERSNL  FB(150 1800)
             ⎪    DEFINE  EMP#         9    5  N
Library      ⎨    DEFINE  EMPNAME     17   20  A
...          ⎪    DEFINE  EMP-COUNT    W    4  N
             ⎩  *

             ⎧  JOB INPUT PERSNL NAME MYPROG
             ⎪    EMP-COUNT = EMP-COUNT + 1
Activities   ⎨    PRINT REPORT1
...          ⎪  *
             ⎪  REPORT REPORT1
             ⎩    LINE EMP# EMPNAME EMP-COUNT
```

In contrast to the above, this example shows fields defined in the activity section of a program. (The DEFINE keyword is required.)

```
             ⎧  FILE PERSNL  FB(150 1800)
Library      ⎨    SALARY-CODE            134    2 N
...          ⎩  *

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

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.

⇒ **Reading 1 of Chapter 3 ends here.**

- **If you have completed the tutorial in Chapter 2, go on to the beginning of Chapter 4.**
- **If you branched to this chapter from the tutorial in Chapter 2, go back to Lesson 2 in Chapter 2.**

⇒ **Reading 2 of Chapter 3 starts here.**

Defining Static Working Storage

Static working storage fields are fields used for storing accumulated values that are printed at the end of a report or are used to compute some other value(s) at the end of a report or at control breaks, such as averages.

Static working storage fields are defined in your program by placing an S in the position on the DEFINE statement where you normally place the field starting position or a W. For example:

```
AVG-GROSS  S  8  N  2
```

Static working storage fields are necessary because of the way CA-Easytrieve processes reports. In the first reading of Chapter 5, "Activity Section - Input and Output," we discuss the PRINT statement and the process that occurs when reports are either sequenced (by the SEQUENCE statement) or are multiple within one JOB activity (more than one REPORT statement is used).

In both cases, CA-Easytrieve outputs data to an intermediary file called a work file or spool file. (See the diagrams under Printing Reports in Chapter 5, "Activity Section - Input and Output.") Work files do not get formatted into reports (through report definition statements) until they have first been sequenced or until the system printer becomes available.

Due to the use of intermediary work files, two different types of working storage fields are needed. The following discussion provides the differences between them and helps you to understand the need for these two field types.

Static Versus Non-Static

Unlike static working storage fields (type **S**), non-static working storage fields (type **W**) are output to work files for every record in the input file. This is done whenever the non-static working storage field is referenced in a REPORT subactivity. If such a field is used to accumulate values during the processing of an entire file, its value, at the time each record is output to the work file, appears on the record in the work file. If the file is then sequenced, the non-static working storage fields are sequenced along with the rest of the fields on the record. This means that accumulated results *do not* appear, either internally or when printed (if printed), in the order they were accumulated.

Therefore, any calculations based on the value of a non-static working storage field performed at the time of report formatting are likely to produce results which are in error. This is only true for non-static working storage fields (type **W**) used to accumulate values for sequenced reports. For example:

Work File Before SEQUENCE	
SEQUENCE KEY FIELD	W-TYPE ACCUMULATOR FIELD
ZZZ	1
BBB	2
AAA	3
PPP	4
QQQ	5
SSS	6
CCC	7

——>

Work File After SEQUENCE	
SEQUENCE KEY FIELD	W-TYPE ACCUMULATOR FIELD
AAA	3
BBB	2
CCC	7
PPP	4
QQQ	5
SSS	6
ZZZ	1

In the above example, the **W**-type field increments by 1 each time a record is processed. Once the work file has been sequenced and the report is formatted, the value contained in the **W**-type field when last processed (output to the report) is now different than it was prior to sequencing. If you attempted to compute averages, based on this value, your results would be in error.

Static working storage fields are not output to work files. This means they are not affected by sequencing. The last value accumulated into an **S**-type field remains unchanged regardless of what is done to the work file and is therefore suitable for any end-of-report calculations such as averaging. For example:

Report File Before SEQUENCE	
SEQUENCE KEY FIELD	S-TYPE ACCUMULATOR FIELD
ZZZ	1
BBB	2
AAA	3
PPP	4
QQQ	5
SSS	6
CCC	7

——>

Work File Before SEQUENCE	
SEQUENCE KEY FIELD	S-FIELD VALUES NOT PASSED TO WORK FILE
ZZZ	
BBB	
AAA	
PPP	
QQQ	
SSS	
CCC	

The above exhibit illustrates that static working storage fields are not copied to work files and therefore are not sequenced as are non-static (type **W**) fields. The static field shown above contains the value seven (7) at the time any averaging is performed at end-of-report.

Initializing Working Storage Fields

To give working storage fields an initial value at the beginning of your program, use the VALUE option of the DEFINE statement. For example:

```
CURR-MON   W   10   A   VALUE 'JANUARY'
```

Assigns an initial value of JANUARY to the alphanumeric working storage field CURR-MON.

When the value clause is not used, numeric working storage fields are automatically initialized to zeros and alphanumeric working storage fields to blanks.

Redefining a Field

Sometimes it is necessary to break a field into several parts to get the exact information you want. A birth date, for example, might have been entered originally as one field in a record. Now, you want to access this information by either the month, day, or year.

Explicit Redefinition

With CA-Easytrieve, you can explicitly redefine the field in the following manner:

```
DATE-OF-BIRTH      103   6   N
   MONTH           103   2   N
   DAY             105   2   N
   YEAR            107   2   N
```

Explicit redefinition requires the exact starting location of each field. Here is a physical representation of the previously defined field:

```
                DATE-OF-BIRTH

 |_|_|_|_|_|_|0|2|1|0|5|5|_|_|_|_|

                 |   |   |
 position       103 105 107
    in
 record:
```

In this exhibit, the MONTH (02) starts in position 103 and occupies positions 103 and 104. The DAY starts in 105 and occupies positions 105 and 106. Finally, YEAR starts in 107 and occupies 107 and 108.

Overlay Redefinition

You can perform overlay redefinition of a field by including the original *field-name* as the starting location for all subsequent fields in the redefinition. This is especially useful when redefining a working storage field which does not have a numeric starting position. For example:

```
DATE-OF-BIRTH                   W     6   N
   MONTH        DATE-OF-BIRTH          2   N
   DAY          DATE-OF-BIRTH  +2      2   N
   YEAR         DATE-OF-BIRTH  +4      2   N
```

The starting position of the redefining field is designated by using the original field name plus any offset (+2 or +4 in the previous example).

When using overlay redefinition, make sure that the redefining field(s) fits within the storage boundaries of the redefined field.

Implicit Start-location

You can define the *start-location* of a field with an implicitly defined position in the record. Implicitly defining a *start-location* eliminates the need to identify the actual *start-location* of a field. Implicit *start-locations* are most useful when you are creating output files, since output files generally have contiguous field locations.

Use an asterisk in place of the numeric *start-location* when implicitly defining a field. The asterisk implies that the field begins in the next available starting position (highest location defined so far, plus one). For example:

```
EMP#       1    5  N
NAME       *   16  A
FILLER1    *   10  N
ADDRESS    *   39  A
```

defines contiguous fields in a record. Since EMP# begins in position 1, then NAME begins in position 6, FILLER1 in position 22, and ADDRESS in position 32. All locations between 1 and 70 are accounted for.

⇒ **Reading 2 of Chapter 3 ends here.**

- **Please continue with the description of the Assignment Statement, which is Reading 2 of Chapter 4.**

⇒ **Reading 3 of Chapter 3 starts here.**

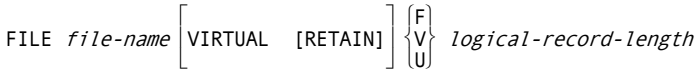
FILE Statement Revisited

In the first reading of this chapter, we discussed the FILE statement and said that it was necessary for describing input and output files. We also said that there are a number of FILE statement parameters. In this reading we discuss two FILE statement parameters that are very useful to you.

Virtual File Manager (VFM)

The VIRTUAL parameter of the FILE statement invokes the virtual file management facility (VFM) of CA-Easytrieve.

Syntax



VFM provides an easy method for establishing temporary work files without special job control or file allocation statements. By using VFM, you can establish your own extract or temporary files, using only CA-Easytrieve keywords.

The FILE keyword and a user-defined file name are required.

Parameters

`VIRTUAL` The VIRTUAL parameter designates that the named file is to be a temporary VFM file. VFM files consist of a dynamically allocated space in memory (64K default). If the allocated space is exhausted, VFM automatically writes the excess data to a single spill area on disk.

`RETAIN` The RETAIN parameter specifies that the VFM file is to remain in memory until the end of the associated CA-Easytrieve execution. If RETAIN is not specified, the VFM file is deleted once it has been read back into your program.

`logical-record-length` CA-Easytrieve requires that you specify a record length for all output files. When specifying record length, you must also specify record type (F, V, or U). Blocksize is not required since VFM files are automatically blocked.

EXIT Parameter

The EXIT parameter on the FILE statement invokes a user routine for every input or output operation performed on the named file. You can use EXIT to access your own user-written routine to convert non-standard data files CA-Easytrieve does not process directly. EXIT is not valid for VFM.

Syntax

```
FILE file-name [EXIT (program-name  +
  [USING (( parm-field-name }... )  [MODIFY] )]  +
         ( parm-literal     )
  [WORKAREA area-length]
```

The EXIT parameter followed by *program-name* indicates the routine or subprogram to be executed.

Parameters

USING USING specifies any parameters to be passed to the exit routine. It is limited to working storage fields, system defined fields, and card literals.

MODIFY The MODIFY option specifies that CA-Easytrieve provides input or output services but that the exit can inspect and modify each record after input and before output.

WORKAREA WORKAREA specifies that CA-Easytrieve sets up a special area of storage to be used as the data buffer for the file. *Area-length* is used to specify the length of the data buffer.

COPY Statement

The COPY statement duplicates the field definitions of a named file. You can copy the field definitions of a given file an unlimited number of times.

Syntax

```
COPY  file-name
```

If you copy the same field name into more than one file and the files are used in the same activity, you must qualify the field when referencing it in your programs or CA-Easytrieve cannot uniquely identify the data reference. You can qualify fields in CA-Easytrieve by preceding them with their file name and a colon. For example, OUTFILE:NAME.

Example of COPY Statement

```
FILE PERSNL  FB(150 1800)
  EMPNAME         17      20 A     HEADING ('EMPLOYEE NAME')
```

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

⇒ **Reading 3 of Chapter 3 ends here.**

- **Please continue with the description of the User Procedures (PROCs), which is Reading 3 of Chapter 4.**

Chapter 4

Activity Section - Processing and Logic

Introduction

The activity section of a CA-Easytrieve program is where all processing logic and report declarations reside. You might say it's where the action is. The activity section can contain PROGRAM, SORT, JOB, and SCREEN activities.

This chapter discusses the JOB, PROGRAM, and SORT activities. Chapter 7, "Activity Section - Screens" gives a detailed description of the SCREEN activities. In this chapter, you'll find:

⇒ **Reading 1**
- CA-Easytrieve JOB statement
- Processing with conditional expressions
- Combined conditions
- Arithmetic calculations

⇒ **Reading 2**
- Assigning values to variables
- Rounding values
- Moving data
- Processing with loops and branches including
- GOTO, DO WHILE, and CASE statements
- STOP statement

⇒ **Reading 3**
- Processing with Procedures, including START and FINISH
- Processing Tables
- Controlling activities with a PROGRAM activity
- Sorting data and the SORT statement

⇒ **Reading 1 of Chapter 4 starts here.**

JOB Activities

JOB Statement

The JOB statement defines and initiates processing activity. It also identifies the name of the automatic input file.

Syntax

```
JOB [INPUT file-name] [NAME job-name]
```

JOB statement parameters can be coded in any order.

Parameters

INPUT The optional INPUT parameter identifies the automatic input to the activity. This means that all input-related logic, such as opening the file, checking for end of file, and reading, are all controlled by CA-Easytrieve.

When you do not specify INPUT, CA-Easytrieve automatically provides an input file. If a SORT activity immediately preceded the current JOB activity, the default input is the output file from that SORT activity. Otherwise, the default input is the first file named in the library section.

file-name The *file-name* identifies the automatic input file(s). It can identify any file defined in the library section of the program eligible for sequential input processing.

NAME *job-name* The optional NAME parameter names the JOB activity and is normally used only for documentation purposes. The *job-name* can:

- Be up to 128 characters in length
- Contain any character other than a delimiter
- Begin with A through Z, 0 through 9, or a national character (#, @, $)
- Not consist of all numeric characters.

The following exhibit shows the location of the JOB statement and the subactivities in a CA-Easytrieve program.

Activity Logic

```
     ** Library **
     *
JOB INPUT PERSNL NAME EXAMPLE
     IF DEPARTMENT = 911 THRU 914 921
          DEDUCTIONS = GROSS - NET
          PRINT EXAMPLE
```

```
        END-IF
      *
Report REPORT EXAMPLE
        SEQUENCE DEPARTMENT NAME
        TITLE 1 'EXAMPLE REPORT'
        LINE 1 EMPNAME DEPARTMENT EMP# GROSS NET DEDUCTIONS
```

You can use the logic subactivity to examine and manipulate data, initiate printed reports, and write data to a file.

You can use the report subactivity to format the desired report.

Conditional Expressions

Data selection and manipulation takes place in the logic section of a CA-Easytrieve program. Logic is coded immediately after the JOB statement.

IF Statement

Processing within a JOB activity can be dependent on the conditional (IF) statements present in the program.

- When an IF statement is present, records read from the input file are processed according to the conditions it states.
- Every IF statement must end with END-IF.

If Statement Syntax

```
IF field-one { EQ  = }   { field-two              }
             { NE  ¬= }  { literal                }
             { GT  >  }  { arithmetic-expression  }
             { GE  >= }
             { LT  <  }
             { LE  <= }

   [statements executed for true IF condition]

[ELSE-IF alternate-expression]

   [statements executed for true ELSE-IF condition]

[ELSE]

   [statements executed for false IF condition]

 END-IF
```

IF Statement Examples

- Comparing the value of a field to a literal:

```
IF DEPT = 910
IF EMPNAME = 'SMITH'
IF AMT GE 500
```

- Comparing two fields:

```
IF DIV = HOLD-DIV
```

- Comparing the value of a field to a series or range of values:

```
IF STATE = 'GA' 'SC' 'TN'
IF CLASS = 'A' THRU 'E'
IF AMT  NE 100 THRU 500
IF DEPT  = 900 940 THRU 950 960 970 THRU 980
```

Arithmetic Operators

These arithmetic operators are valid in conditional statements:

Operators	Meaning
EQ =	Equal To
NE Ø=	Not Equal To
GT >	Greater Than
GE >=	Greater Than or Equal To
LT <	Less Than
LE <=	Less Than or Equal To

Parameters

IF/ELSE ELSE directs CA-Easytrieve to perform alternative processing when the condition, established by the IF statement, is not met.

- For true IFs, all commands up to the ELSE (or END-IF if no ELSE is present) are executed.
- For false IFs, commands between ELSE and END-IF are executed.
- Following END-IF, processing continues regardless of the result of the IF.

IF/ELSE Example

```
IF DIVISION = 'A' THRU 'L'
  DEDUCTIONS = GROSS * .15
ELSE
  DEDUCTIONS = GROSS * .18
END-IF
```

In the above example, records with a DIVISION field containing values in the A through L range are processed according to the statement between the IF and ELSE statements (DEDUCTIONS = GROSS * .15). For records with DIVISION not in the range A through L, the statement following ELSE (DEDUCTIONS = GROSS * .18) is executed. END-IF signifies the end of the condition.

In words, we could restate the condition in the above example something like this, “For divisions A through L, deductions are equal to 15 percent of the gross; for all other divisions deductions are equal to 18 percent of the gross.”

ELSE-IF ELSE-IF is optional and identifies a conditional expression to be tested when the previous conditional expression is false. ELSE-IFs permit multiple conditions to be nested without requiring an END-IF for each condition. You can code as many ELSE-IFs as necessary.

ELSE-IF Example

```
IF DIVISION = 'A' THRU 'L'
  DEDUCTIONS = GROSS * .15
ELSE-IF DIVISION = 'M' THRU 'R'
  DEDUCTIONS = GROSS * .2
ELSE
  DEDUCTIONS = GROSS * .18
END-IF
```

Special IF Statements

Use Special IF statements to check the integrity of the data in your files.

Syntax

```
                          ⎧ ALPHABETIC ⎫
                          ⎪ NULL       ⎪
                          ⎪ NUMERIC    ⎪
IF  field-name [NOT]      ⎨ SPACE      ⎬
                          ⎪ SPACES     ⎪
                          ⎪ ZERO       ⎪
                          ⎪ ZEROS      ⎪
                          ⎩ ZEROES     ⎭
```

Special IF statement keywords check for the following conditions:

Keyword	Condition
ALPHABETIC	Value containing characters A through Z and blank spaces.
NULL	No current value.
NUMERIC	Value containing digits 0 through 9.
SPACE SPACES	Value containing all blank spaces.
ZERO ZEROS ZEROES	Value containing all zeros (0).

Special IF Examples

This statement is true ...	for this condition ...
`IF AMT NOT NUMERIC`	AMT does not contain all digits

This statement is true ...	for this condition ...
`IF NAME SPACES`	NAME contains all spaces
`IF STATE ALPHABETIC`	STATE contains all letters and spaces
`IF AMT-DUE ZERO`	AMT-DUE contains all zeros

Combining Conditional Expressions

Conditional expressions can be compounded by combining them through the logical connectors AND and OR. For example, if you need to determine a value based on two conditions, you can connect the conditions with a logical connector:

```
IF DIVISION = 'A' THRU 'L' AND AMOUNT GE 15
```

This statement is true when DIVISION is equal to a letter in the range A through L and when AMOUNT is also greater than or equal to 15. Both conditions must be true for the entire statement to be true. The following statement uses the OR connector:

```
IF DIVISION = 'A' THRU 'L' OR AMOUNT GE 15
```

This statement is true when DIVISION is equal to a letter in the range A through L or when AMOUNT is greater than or equal to 15 or when both conditions are true. Either one or both of the conditions can be true to make the entire statement true.

When used together in the same statement, conditions connected by AND are examined before conditions connected by OR, for example:

```
IF DIVISION = 'A' AND AMOUNT GE 15 OR STATE = 'GA'
```

In the previous statement, CA-Easytrieve examines the portion "DIVISION = 'A' AND AMOUNT GE 15" first. If both sides of the AND in that portion are found to be true then the entire statement is true. If not, then the portion "OR STATE = 'GA'" is examined and if found to be true, the entire statement is still true. If conditions on both sides of the OR are false, then the entire statement is false.

The following table helps you visualize the concept of logical connectors. The assumptions for the table are:

```
DIVISION = A
AMOUNT = 15
STATE = GA
```

The following IF statement ...	is ...
`IF DIVISION = 'A' AND AMOUNT GE 15 OR STATE = 'GA'`	TRUE
`IF DIVISION = 'A' AND AMOUNT = 14 OR STATE = 'FL'`	FALSE

The following IF statement ...	is ...
`IF DIVISION = 'A' OR AMOUNT = 15 AND STATE = 'FL'`	TRUE
`IF DIVISION = 'B' AND AMOUNT = 15 AND STATE = 'FL'`	FALSE
`IF (DIVISION = 'A' OR AMOUNT = 15) AND STATE = 'FL'`	FALSE

Note: Inserting parentheses around a set of conditions can alter the outcome of the statement. Remember these three rules:

- All conditional expressions are considered one statement.
- AND statements are evaluated before ORs.
- Parentheses may alter the normal order of evaluation.

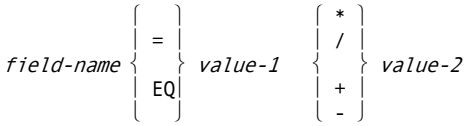
Calculations

There are four arithmetic operations in CA-Easytrieve:

* multiplication
/ division
\+ addition
\- subtraction

Multiplication and division are performed before addition and subtraction in order from left to right. There must be a space before and after the arithmetic operators.

Syntax



Parentheses in Calculations

Parentheses can be used to override the normal order of operation. Operations contained in parentheses are performed first, for example:

```
RESULTS = GROSS - AMT * 1.3
```

is the same as:

```
RESULT = GROSS - (AMT * 1.3)
```

but different from:

```
RESULT = (GROSS - AMT) * 1.3
```

You can "nest" parentheses to further alter the order of operation. Operation proceeds from the innermost set of parentheses to the outermost:

```
RESULT = 1.3 * (GROSS - (AMT + DEDUCT))
```

In the above example, AMT and DEDUCT are added before being subtracted from GROSS. After subtraction, the difference is multiplied by 1.3 and the product of this is assigned to the RESULT field.

⇒ **Reading 1 of Chapter 4 ends here.**

- **If you have completed the tutorial in Chapter 2, go on to the beginning of Chapter 5.**
- **If you branched to this chapter from the tutorial in Chapter 2, go back to Lesson 3 in Chapter 2.**

⇒ **Reading 2 of Chapter 4 starts here.**

Assignment Statement

The Assignment statement establishes a value in a field by copying the value from another field or literal. The value on the right of the equal sign is copied to the field on the left of the equal sign. The Assignment statement also accomplishes data conversion, such as packing or unpacking numeric data.

Syntax

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

Simple Assignment Examples

```
HOLD-DIV = DIV
DEPT-NAME = 'ACCOUNTING DEPT'
RATE = 1.1
```

MOVE Statement

Use the MOVE statement to transfer data from one location to another. MOVE is useful for moving data without conversion and for moving character strings with variable lengths.

- You can move a field or a literal to a field, or move a file to a file.
- A sending field longer than a receiving field is truncated on the right.
- A receiving field longer than the sending field is padded on the right with spaces or an alternate fill character.

- Spaces or zeros can be moved to one or many fields.

The MOVE statement has two formats.

MOVE Format 1

Syntax

```
MOVE {send-file-name   } [send-length] TO {receive-file-name   } +
     {send-field-name  }                  {receive-field-name  }
     {send-literal     }

[receive-length]     [FILL fill-character]
```

When you specify Format 1, data moves from one field to another, filling with spaces or a specified fill character on the right. The FILL parameter enables you to place specified characters in the unused spaces of the new field (the default is blank spaces).

Remember, that the MOVE statement does not convert data as it is moved. Use the Assignment statement to convert the data from one field's data type to another's.

Examples

```
MOVE NAME 20 TO HOLD-NAME
```

Moves the first 20 characters of the NAME field to the HOLD-NAME field.

```
MOVE NAME CTR TO HOLD-NAME FILL '*'
```

A numeric length for the sending field (NAME) is replaced here by a field name CTR. CTR contains a numeric value that determines the number of characters moved to HOLD-NAME. Any remaining spaces after the move (assuming the sending field is smaller than the receiving field) are filled with asterisks.

MOVE Format 2

Syntax

```
MOVE  { NULL   }      TO   field-name-1 field-name-n
      { SPACE  }
      { SPACES }
      { ZERO   }
      { ZEROS  }
      { ZEROES }
```

You can use Format 2 to initialize the receiving field.

Example

```
MOVE SPACES TO NAME, HOLD-NAME, HOLD-DIV
```

Fills all of the named fields with blank spaces.

MOVE LIKE

MOVE LIKE moves the contents of fields in one file to identically named fields in another file.

Syntax

```
MOVE LIKE  file-name-1  TO  file-name-2
```

It is important to understand that the MOVE LIKE statement creates assignments of each LIKE field. These assignments perform data conversions, if necessary.

Example

```
FILE  INFILE1
  EMPNAME 17  20  A
  DEPT    98   3  N
  AMT     90   4  P  2
FILE OUTFIL1
  AMT      1   7  N  2
  EMPNAME  8  11  A
JOB INPUT INFILE1 NAME MOVE-LIKE-EXAMPLE
  ** Logic **
*
MOVE LIKE  INFILE1  TO  OUTFIL1
  ** Logic **
```

In the above example, the EMPNAME field of INFILE1 is moved to the EMPNAME field of OUTFIL1 where the last nine characters are truncated. The AMT field of INFILE1 is moved to the AMT field of OUTFIL1 where it is converted to numeric format from packed decimal format.

DO/END-DO Statements

Use the DO and END-DO statements to provide a controlled loop for repetitive program logic.

Syntax

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

    ** Logic **

END-DO
```

Parameters

```
[WHILE]
{     }
[UNTIL]
```

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

conditional-expression Specify the condition that is the basis for the continuing execution of the loop. Conditional expressions follow the rules of IF statements.

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

DO WHILE Example

```
JOB INPUT PERSNL NAME DO-EX-1
CTR = 0
DO WHILE CTR LT 10
  CTR = CTR + 1
  ** Logic **
END-DO
```

The above DO WHILE statement causes "CTR = CTR + 1" to repeat until CTR is equal to 10. At that point, control transfers to the first statement after the END-DO statement.

DO UNTIL Example

```
JOB INPUT PERSNL NAME DO-EX-2
CTR = 0
DO UNTIL CTR GE 10
  CTR = CTR + 1
  ** Logic **
END-DO
```

The previous DO UNTIL statement causes "CTR = CTR + 1" to execute the logic once then repeat until CTR is equal to 10. At that point, control transfers to the first statement after the END-DO statement.

As you can see by the above examples, you can use the WHILE and UNTIL parameters of the DO statement to perform identical tasks. The rule of thumb to follow when trying to determine which parameter to use is to use UNTIL if you want to be sure the logic (CA-Easytrieve statements) is executed at least once. The UNTIL parameter causes CA-Easytrieve to perform the logic then evaluate the conditional expression.

Use WHILE if you do not want the logic executed. The WHILE parameter causes CA-Easytrieve to evaluate the conditional expression and perform the logic only if the condition is true.

DO Nesting Example

You can nest DO statements. (The inner logic loop must be completely within the outer logic loop.)

```
JOB INPUT PAYROLL NAME DO-EX-3
  CTR1 = 0
  DO WHILE CTR1 LT 10
    CTR2 = 0
      DO WHILE CTR2 LT 5
```

```
      CTR2 = CTR2 + 1
                                  Inner
      ** Logic **                 Loop
                                          Outer
    END-DO                                 Loop
  CTR1 = CTR1 + 1

** Logic **

END-DO
```

In the above example, the inner DO WHILE loop executes five times for each single execution of the outer loop. When CTR1 is equal to 10, control is passed to the first statement following the outer END-DO statement.

CASE and END-CASE Statements

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

Syntax

```
CASE   field-name

  WHEN compare-literal-1 [THRU range-literal-1]
    (statements)

  WHEN compare-literal-n [THRU range-literal-n]
    (statements)

  [OTHERWISE]
    (statements)

END-CASE
```

Parameters

field-name Specifies a field that contains a value that is compared to the values represented by *compare-literal* [THRU *range-literal*]. *Field-name* can be a field of any type except a varying length alphanumeric field. If *field-name* is alphanumeric, it must be 254 or fewer bytes in length. If *field-name* is numeric, it must have zero or no decimal places.

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

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

When *field-name* is alphanumeric, *compare-literal* and *range-literal* must also be alphanumeric and must be equal in length to *field-name*. When *field-name* is defined as a numeric data type, *compare-literal* and *range-literal* must also be numeric and must not have any decimal places. Numeric literals need not be equal in length to *field-name*.

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

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

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

Nesting CASE Statements

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

Example

The following example uses CASE to compare the value in JOB-CATEGORY to the range specified in the WHEN clauses and calculate Christmas bonuses, based on that value.

```
FILE PERSNL FB (150 1800)
  EMPNAME        17  8  A
  EMP#            9  5  N
  DEPT           98  3  N
  GROSS          94  4  P  2
XMAS-BONUS        W  4  P  2
JOB-CATEGORY    132  2  N  2
```

Continued

Continued

```
JOB INPUT PERSNL NAME COMPUTE-XMAS-BONUS
  CASE JOB-CATEGORY
    WHEN 1 THRU 29
      XMAS-BONUS = PAY-GROSS * 1.03
    WHEN 30 THRU 59
      XMAS-BONUS = PAY-GROSS * 1.05
    OTHERWISE
      XMAS-BONUS = PAY-GROSS * 1.07
END-CASE
PRINT RPT
REPORT RPT
LINE NAME-LAST XMAS-BONUS
```

GOTO Statement

You use the GOTO statement to branch out of the normal top-to-bottom logic flow in a program.

Syntax

```
⎧GOTO  ⎫  ⎧label ⎫
⎨      ⎬  ⎨JOB   ⎬
⎩GO TO ⎭  ⎩SCREEN⎭
```

This statement directs program control to another area in the program. CA-Easytrieve accepts either GOTO or GO TO.

Parameters

GOTO *label* *Label* refers to a statement label. GOTO *label* transfers control immediately to the first statement following the named statement label. The statement label can be anywhere in the same activity or procedure. A statement label can:

- Be up to 128 characters in length
- Contain any character other than a delimiter
- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

GOTO JOB Transfers control to the top of the current JOB activity. This is useful to stop specific records from further processing.

GOTO SCREEN Transfers control to the top of the current SCREEN activity. See Chapter 7, "Activity Section - Screens" for more information.

Example

```
JOB INPUT PERSNL NAME DIV-LIST  <-------┐  Transfers
  IF DIV = 'A'                        |   Control
    GOTO JOB ---------------------------┘
  END-IF
  IF DIV = 'B'
    GOTO CHECK-REG-ROUTINE    ----------┐
  END-IF                                |
                                        |  Transfers
     ** Logic **                        |  Control
                                        |
  CHECK-REG-ROUTINE    <----------------┘

     ** More Logic **
```

STOP Statement

A STOP statement enables you to terminate an activity.

Syntax

```
STOP   [EXECUTE]
```

- STOP ends the current JOB or SORT activity, completes the report processing for the activity, if any, and then goes on to the next JOB or SORT activity if one exists. A FINISH procedure (if one is present) is still executed before going on to the next JOB or SORT activity.
- STOP EXECUTE immediately terminates all CA-Easytrieve execution.

Example

```
IF AMT NOT NUMERIC
  STOP
END-IF
```

⇒ **Reading 2 of Chapter 4 ends here.**

- **Please continue with the description of User Controlled Input and Output, which is Reading 2 of Chapter 5.**

⇒ **Reading 3 of Chapter 4 starts here.**

User Procedures (PROCs)

A user procedure (also called PROC for short) is a group of user-written CA-Easytrieve statements designed to accomplish some task. PROCs are useful when developing structured programs which modularize discrete and repetitive tasks.

Invocation Syntax

PROCs are invoked by using the PERFORM statement which has the following format:

```
PERFORM  proc-name
```

Parameters

proc-name Specifies the name of a user-defined procedure located at the end of the activity in which it is PERFORMed. *Proc-name* can:

- Be up to 128 characters in length
- Contain any character other than a delimiter
- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

As mentioned earlier, procedures are discrete modules of program code which perform a task.

Procedure Syntax

When coded, procedures must have this format:

```
   proc-name. PROC

** Procedure Logic **

   END-PROC
```

Parameters

PROC The PROC keyword must follow the *proc-name* separated by a period and a space. *Proc-name* is the same name as on the PERFORM statement.

END-PROC Every PROC must have an END-PROC which marks the end of the procedure. At END-PROC, control is returned to the statement following the PERFORM statement that invoked the PROC.

Procedure Example

The following example performs two simple procedures, based on the value of a field named CODE.

```
IF CODE = 1
  PERFORM CODE1-RTN
ELSE
  PERFORM CODE2-RTN
END-IF

** Logic **

CODE1-RTN. PROC
  ORDER = 'NO'
END-PROC
CODE2-RTN. PROC
  ORDER = 'YES'
END-PROC
```

Nesting PROCs

A PERFORM statement within a procedure can invoke another procedure. For example:

```
IF DEPT = 911
  PERFORM PROCA
END-IF

        ** Logic **

  PROCA. PROC
    IF ST = 'NY'
      PERFORM PROCB
    ELSE
      TAX = GROSS * .05
    END-IF
  END-PROC
  PROCB. PROC
    TAX = GROSS * .1
  END-PROC
```

START/FINISH Procedures

You use the optional START and FINISH parameters of the JOB statement to automatically incorporate procedures into processing activities.

Syntax

The format for invoking these procedures is as follows:

```
JOB INPUT file-name [NAME job-name]      +
    [START start-proc-name] [FINISH finish-proc-name]
```

Parameters

START *start-proc-name* START procedures are used to execute routines prior to execution of the logic in the body of the JOB activity.

- The procedure is invoked automatically after the file is opened but prior to reading the first input record.
- A typical START procedure might initialize working storage fields or establish a position in a keyed sequenced file (see POINT statement in Chapter 5, "Activity Section - Input and Output").

FINISH *finish-proc-name* FINISH procedures are used to identify a procedure to be executed during the normal termination of the JOB activity.

- The procedure is invoked after the last input record is processed but before any files are closed.
- A typical FINISH procedure displays control information accumulated during execution of the JOB activity.
- CA-Easytrieve still executes FINISH *procs* if a STOP statement is encountered during the course of the program but not if a STOP EXECUTE is encountered.

Processing Tables

A table is a collection of uniform data records in a form suitable for quick reference.

Much like books in a library, a table has two components; an identifier that helps you find the information you are looking for (analogous to a card catalog number) and the information you are looking for (a book). With tables however, the identifier is called a *search argument;* the information you are after is called the *description.* Each entry in a table must consist of:

- A search argument that uniquely identifies the entry. This is defined as a field with the name ARG after the FILE statement.

- A description (the data) associated with the search argument. This is defined as a field with the name DESC after the FILE statement.

Your objective is to obtain the description from a table, based on the search argument. Rules governing the processing of search arguments are as follows:

- A table file must be arranged in ascending order by search argument.
- No duplicate search arguments can be placed in the file.
- You can use any number of tables in a job.
- A minimum of three entries is required in a table.

The following example shows a table with search arguments and descriptions. The argument is a numeric code used to look up a descriptive state name.

```
 ARG          DESC

 01           ALABAMA
 02           ALASKA
 03           ARIZONA
 ...
 47           WASHINGTON
 48           WEST VIRGINIA
 49           WISCONSIN
 50           WYOMING
```

Creation of Table Files

The creation of tables involves the inclusion of certain parameters on the CA-Easytrieve FILE statement:

Syntax

```
                        ┌INSTREAM          ┐
FILE file-name TABLE    │                  │
                        └max-table-entries┘
```

Parameters

TABLE The TABLE parameter of the FILE statement declares that the file is the object of a CA-Easytrieve SEARCH statement which is used to access tables. Tables can be either *external* (stored in a file outside your program) or *instream* (data is included within your program). External table files must be sequentially accessible.

INSTREAM Denotes that the table file data is within your program. Such data immediately follows the file description after the ARG and DESC field definitions.

max-table-entries Specifies the maximum number of entries (records) in an external table. Specify a value here only if the number of entries is greater than the maximum stored in the Site Options Table.

Instream Table Example

The word ENDTABLE must be the last entry in an instream table and must be coded in columns 1 through 8.

```
FILE   STATTBL   TABLE  INSTREAM
  ARG    1   2  N
  DESC   4  15  A
01 ALABAMA
02 ALASKA
03 ARIZONA
 ...
47 WASHINGTON
48 WEST VIRGINIA
49 WISCONSIN
50 WYOMING
ENDTABLE
```

The previous example defines a table of state names which can now be looked up according to a two-digit code.

Accessing Table Files

SEARCH Statement

The SEARCH statement is used to perform a search of a table. SEARCH can be:

- Coded any place within a JOB, PROGRAM, or SCREEN activity
- Issued any number of times against any number of tables.

Syntax

The SEARCH statement has this format:

```
SEARCH file-name    WITH search-field  GIVING result-field
```

file-name The name of the table that appears on the FILE statement.

search-field The name of a field which contains a value that is compared to the search argument. It must be the same length and type as the search argument (ARG).

result-field The name of a field into which the description is placed if a match exists between *search-field* and the search argument. It must be the same length and type as the description (DESC).

Testing for a Match

After using the SEARCH statement, you can test to determine whether a match was found between the *search-field* and the search argument by using a special IF statement.

Syntax

The IF statement has this format:

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

External Table Example

```
FILE PERSNL
  EMPNAME           17  8 A
  STATE             69  2 A
  ZIP               71  5 N
  GROSS-PAY         94  4 P 2
  POST-OFFICE-DESC   W 20 A
FILE ZIPTABLE TABLE 5000
  ARG      1  5   N
  DESC     7  20  A
JOB INPUT PERSNL NAME TABLE-SEARCH
  IF STATE = 'DC' 'IL'
    SEARCH ZIPTABLE WITH ZIP GIVING POST-OFFICE-DESC
    IF NOT ZIPTABLE
       POST-OFFICE-DESC = 'BAD ZIP CODE FOUND'
    END-IF
    PRINT STATE-REPORT
  END-IF
REPORT STATE-REPORT
  SEQUENCE STATE
  CONTROL STATE
  TITLE 1 'REPORT OF EMPLOYEE SALARIES BY STATE'
  LINE 1 STATE EMPNAME GROSS-PAY ZIP POST-OFFICE-DESC
```

SORT Activities

SORT Statement

SORT is a separate activity (outside the activity of the JOB statement) that sequences an input file in alphabetical or numerical order based on fields specified as keys. You can sort on as many fields as your system allows. (The SORT activity uses the sort utility provided by your system.)

Syntax

```
SORT input-file-name  TO  sorted-file-name   +
   USING (sort-key-field-name [D] ...)       +
   NAME sort-name
```

Parameters

input-file-name The input file to be sorted.

sorted-file-name The output file.

USING *sort-key-field-name* Identifies those fields from *input-file-name* that you use as sort keys. Keys are specified in "major to minor" order. This dictates how information is sorted. For example, you could sort a file of employee records by region, and then by location under region and then by department under location. Region would be the major sort key, location would be minor, and department would be more minor.

D Optionally sorts the field contents in descending order (ascending order is the default).

NAME *sort-name* Like NAME on the JOB statement, this parameter identifies the sort activity and is normally used for documentation purposes only. *Sort-name* can:

- Be up to 128 characters in length
- Contain any character other than a delimiter
- Begin with A-Z, 0-9, or a national character (#, @, $)
- Not consist of all numeric characters.

Sort Example

```
FILE PERSNL FB(150 1800)
  EMPNAME     1  10  A
  DEPT       11   5  N
  GROSS-PAY  16   4  P 2
FILE PAYSORT FB(150 1800)
SORT PERSNL TO PAYSORT   +
  USING (DEPT GROSS-PAY)  +
  NAME SORT-EXAMPLE-1
```

The above SORT activity sorts the file PERSNL in ascending order, first by DEPT and then by GROSS-PAY under DEPT. This produces a file containing records in order by department and records with LIKE departments in order by gross pay.

SORT Procedures

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

Syntax

A SORT procedure is executed through the BEFORE parameter of the SORT statement:

```
SORT  input-file-name  TO  sorted-file-name  +
```

```
USING  (sort-key-field-name  [D] ... )      +

NAME  sort-name                    +

[BEFORE proc-name]
```

- A SORT procedure must immediately follow the SORT statement.
- You invoke a SORT procedure with the BEFORE parameter.
- The SORT procedure executes for each record from *input-file-name* prior to passing the record to the sort.

Parameters

BEFORE *proc-name* Identifies the user-defined procedure you want to execute. CA-Easytrieve supplies input records to your SORT procedure one at a time. If a BEFORE procedure is used, the SELECT statement must be executed for each record that you want to sort.

- You must execute a SELECT statement for each record that you want returned to the output file.
- A SELECTed record outputs only once, even if SELECTed more than once in the procedure.
- Any record not SELECTed does not go to the sorted file.

Select Syntax

```
SELECT
```

Sort Procedure Example

This example illustrates the use of the SORT activity and SORT procedures.

```
FILE PERSNL FB(150 1800)
  EMPNAME           1   10   A
  DEPT             11    5   N
  GROSS-PAY 16      4        P 2
FILE PAYSORT F(19) VIRTUAL
  SORT-NAME         1   10   A
  SORT-DEPT        11    5   N
  SORT-GROSS-PAY   16    4   P 2
JOB INPUT PERSNL NAME ACT-1
  PRINT RPT1
REPORT RPT1
  LINE 1 NAME DEPT GROSS-PAY
SORT PERSNL TO PAYSORT USING (DEPT GROSS-PAY D) +
  BEFORE SELECT-REC NAME SORT-ACTIVITY
  SELECT-REC. PROC
    IF GROSS-PAY  GE  500
      SELECT
    END-IF
  END-PROC
```

In the above example, SELECT-REC is the name of the SORT procedure. The procedure causes only those records with a gross pay of greater than or equal to 500 to be selected for sorting.

PROGRAM Activities

A PROGRAM activity can be used for simple processing activities or to control the execution of JOB, SORT, and SCREEN activities.

Simple PROGRAM Example

The following example illustrates the use of a PROGRAM activity where you merely need to perform an arithmetic computation, display the results, then stop processing.

```
DEFINE RESULT W 4 P 2
PROGRAM NAME COMPUTE
  RESULT = (2354.54 * 6) /3.8
  DISPLAY THE RESULT IS  RESULT
```

Controlling Other Activities

A PROGRAM activity can be used to conditionally initiate JOB, SORT, and SCREEN activities.

```
PROGRAM NAME PROCESSOR
  IF SYSTIME = 09:00:00 THRU 17:00:00
    EXECUTE PRIME-TIME-JOB
  ELSE
    EXECUTE OFF-TIME-JOB
  END-IF
JOB NAME PRIME-TIME-JOB
  ** Logic **
JOB NAME OFF-TIME-JOB
  ** Logic **
```

The PROGRAM activity, in the above example, controls which JOB activity is executed based on the time of day. If a PROGRAM activity had not been specified, the JOB activities would have been executed sequentially from the first JOB activity.

EXECUTE Statement

The EXECUTE statement invokes a JOB, SORT, or SCREEN activity from either a PROGRAM or SCREEN activity. The EXECUTE statement transfers control to an activity. After the activity is executed, control returns to the next executable statement following the EXECUTE.

Note: You cannot invoke a JOB, SORT, or SCREEN activity within a JOB or SORT activity.

EXECUTE statements within a SCREEN activity can invoke other activities. This is called activity nesting.

Syntax

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

Parameters

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

Name the JOB, SORT, or SCREEN activity to be executed.

⇒ **Reading 3 of Chapter 4 ends here.**

- **Please continue with the description of the POINT Statement, which is Reading 3 of Chapter 5.**

Chapter

5

Activity Section - Input and Output

Introduction

In CA-Easytrieve, file input/output can be either controlled by CA-Easytrieve (automatic) or controlled by you (user controlled). There are several statements available for providing input and output under a variety of conditions. All input and output occurs in the activity section of your programs.

In this chapter, you'll find:

⇒ **Reading 1**

- Automatic input using the JOB statement
- Report output using the PRINT statement

⇒ **Reading 2**

- User-controlled input/output of sequential access files, including the use of the DISPLAY, GET, and PUT statements

⇒ **Reading 3**

- Using the POINT statement to establish a starting position for sequential processing of a keyed file
- Programmer-controlled input/output of randomly accessed files, including the use of the READ and WRITE statements

⇒ **Reading 1 of Chapter 5 starts here.**

Automatic Input and Output

CA-Easytrieve gives you the option of letting it take care of input and output for you. All of the usual "housekeeping" considerations like opening and closing files, checking for end of file, issuing input and output statements in a loop, can be taken care of automatically.

Automatic Input with the JOB Statement

The JOB statement lets you identify a file for automatic input to the JOB activity. All you have to do is specify the file name after the word INPUT; CA-Easytrieve takes care of all the rest.

Syntax

Here is how the JOB statement looks with automatic input:

```
JOB [INPUT  file-name]
```

When you specify INPUT and a file name, the records of that file are automatically made available to the logic in your JOB activity section. However, there are some implied statements being executed which you don't see in your program. Here are the steps actually taken when the JOB statement is executed with automatic input.

```
IF THERE IS A START PROCEDURE
  THEN PERFORM THE START PROCEDURE
END-IF
OPEN FILE(S)
RETRIEVE THE INPUT
IF NO MORE INPUT
  IF THERE IS A FINISH PROCEDURE
   THEN PERFORM THE FINISH PROCEDURE
  END-IF
  WRAP UP THE REPORTS
  GO TO THE NEXT ACTIVITY
ELSE
  PERFORM LOGIC ACTIVITIES
END-IF
GOTO
```

You can leave the INPUT parameter off the JOB statement. If you do, CA-Easytrieve provides the automatic input by getting it from either the first file described in your library section or the output of a directly previous SORT, if any.

Printing Reports

Printing of reports is initiated with the CA-Easytrieve PRINT statement which looks like this:

Syntax

```
PRINT [report-name]
```

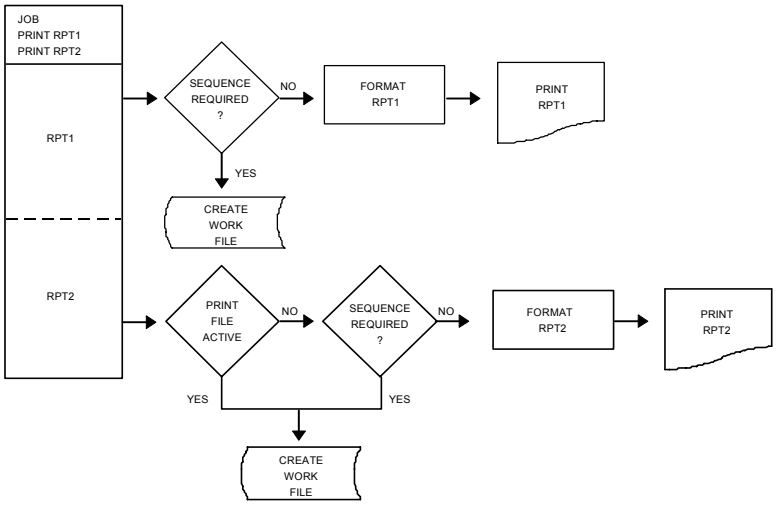
PRINT is not completely automatic in that it does permit you a certain amount of control. You can execute PRINT anywhere in your JOB activity logic and you can use conditional logic to determine when it should execute. But, once PRINT is executed, it activates the designated report declaration and takes care of all output considerations automatically.

In most cases however, your reports do not go directly to a printer (through a print file). Rather, they go to a CA-Easytrieve work file (sometimes called a spool file). Work files are necessary in two cases:

- When the printer (print file) is already activated by a previous report of the same JOB activity. (Multiple reports directed to the same printer).
- When a report requires sequencing (specifically, a SEQUENCE statement is present).

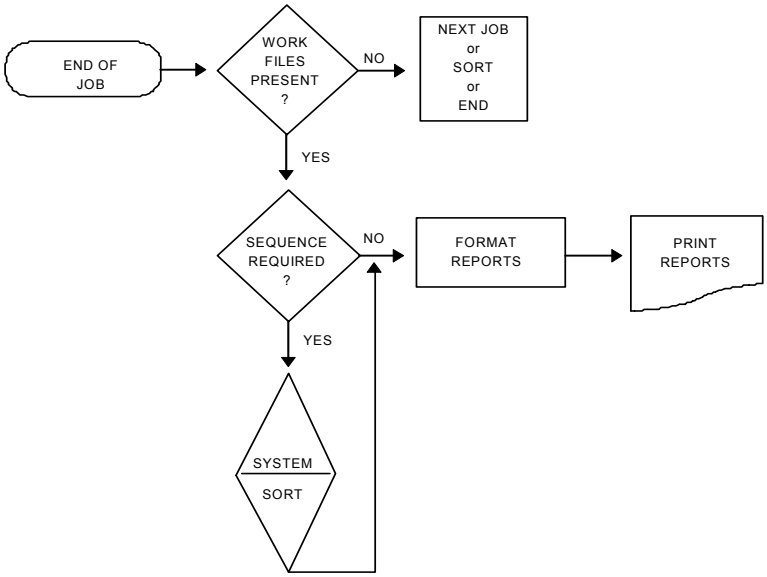
PRINT Statement Execution

The following exhibit illustrates how the PRINT statement is executed for reports going to a single printer.



Work File Processing

If work files are created, as presented in the previous exhibit, they are held until the end of the job activity. At end of job, they are processed as shown in the next exhibit.



⇒ **Reading 1 of Chapter 5 ends here.**

- **If you have completed the tutorial in Chapter 2, go on to the beginning of Chapter 6.**
- **If you branched to this chapter from the tutorial in Chapter 2, go back to Lesson 4 in Chapter 2.**

⇒ **Reading 2 of Chapter 5 starts here.**

User Controlled Input and Output

Sequential File Processing

CA-Easytrieve provides you with three statements for sequentially processing files. The following table lists these statements and their general purpose:

Statement	Purpose
DISPLAY	Normally used to display data to your system output device

Statement	Purpose
GET	Used for sequential retrieval of input file records
PUT	Used for sequential output of records to an output file

DISPLAY Statement

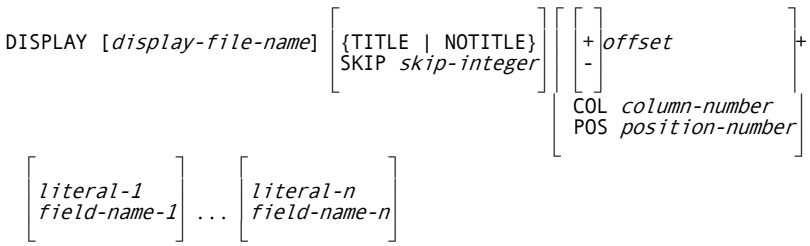
A DISPLAY statement sends data to a specified output file or output device. DISPLAY is commonly used:

- For error messages
- For highlighting reports
- For hexadecimal display of selected information.

If DISPLAY is used in the logic portion of the JOB activity and output is to a report, the lines to be DISPLAYed are interspersed throughout the report in an unSEQUENCEd report or printed at the beginning of a SEQUENCEd report (before the first title). When DISPLAY is used in report procedures, you are only permitted to display to the system output device (not to a data file). The DISPLAY statement has two different formats.

DISPLAY Format 1

Syntax



Parameters

display-file-name When you specify *display-file-name*, CA-Easytrieve prints data to the named file. If you do not specify *display-file-name*, the default is SYSPRINT.

TITLE | NOTITLE The TITLE option specifies that a skip to a new page occurs before the data is printed. Any titles and headings are also produced. NOTITLE specifies that a skip to a new page occurs but titles and headings are not produced.

SKIP *skip-integer* The SKIP option specifies that the designated number of lines are skipped before the data is printed.

offset Coding a positive or negative *offset* modifies the horizontal spacing between display items.

COL *column-number* The COL *column-number* option specifies the print column number where CA-Easytrieve places the next display item.

POS *position-number* When used in report procedures, the *POS position-number* option causes the next display item to be positioned under the corresponding position on the LINE 01 statement.

literal-1,n or ***field-name-1,n*** Code *literals or field-names* in the order you want them to appear on the printed line.

DISPLAY Examples 1

```
DISPLAY SKIP 2 '**RECORD NOT FOUND FOR KEY' +2 SSN

DISPLAY ERRFILE 'THIS REPORT IS FOR ERRORS +
        THAT WERE FOUND IN THE EDIT PHASE.'
```

DISPLAY Format 2

Syntax

```
DISPLAY [display-file-name] [{TITLE | NOTITLE}] HEX [file-name]
                            [SKIP skip-integer]     [field-name]
```

In this format, CA-Easytrieve produces a hexadecimal and character dump of the current record or the specified *field-name*. The parameters, other than HEX, operate the same as in Format 1.

DISPLAY Example 2

```
DISPLAY HEX NAME
```

produces:

```
CHAR WIMN
ZONE ECDD4444444444444444
 NUMR 69450000000000000000
      1...5...10...15...20
```

GET Statement

The GET statement retrieves the next record of the named file into the file input area.

Syntax

```
GET   file-name
```

Parameter

file-name Identifies the input file defined in the library section.

Note: You must test for end-of-file (EOF) when using the GET command.

GET Example

```
FILE MASTER FB(150 1800)
  EMP#     9    5   N
  EMPNAME 17   16   A
  GROSS   94    4   P 2
JOB INPUT NULL NAME READ-SEQ-MAN
GET MASTER
  IF EOF MASTER
     STOP
  END-IF
  IF GROSS > 500
     PRINT RPT1
  END-IF
REPORT RPT1
  LINE 1 EMP# EMPNAME GROSS
```

You cannot use GET for an automatic input file. To inhibit automatic input, specify INPUT NULL on the JOB statement. For example:

```
JOB  INPUT NULL
```

You might GET a secondary file while automatically accessing a primary file.

PUT Statement

The PUT statement outputs to a file sequentially.

Syntax

```
PUT  output-file-name [FROM input-file-name]
```

Parameters

output-file-name Identifies a file defined in the library section to which you are writing data.

FROM *input-file-name* Using the FROM option is like performing a MOVE of data from *input-file-name* to *output-file-name* before performing the PUT.

PUT Example 1

```
FILE PERSNL FB(150 1800)
  EMP#          9   5 N
  EMPNAME      17  16 A
  GROSS        94   4 P 2
FILE NEWPAY2 F(20)
  EMPNAME       1  16 A
  GROSS        17   4 P 2
JOB INPUT PERSNL NAME PUT-EXAMPLE
  ** Logic **
MOVE LIKE PERSNL TO NEWPAY2
PUT NEWPAY2
```

PUT Example 2

```
FILE MASTER FB(150 1800)
  EMP#     9   5   N
  EMPNAME 17  16   A
  GROSS   94   4   P 2
FILE OUTMAST FB(150 1800)
JOB INPUT MASTER NAME CREATE-SEQ
  IF GROSS > 500
*
    PUT OUTMAST FROM MASTER
*
  END-IF
```

⇒ **Reading 2 of Chapter 5 ends here.**

- **Please continue with the description of Label Reports, which is Reading 2 of Chapter 6.**

⇒ **Reading 3 of Chapter 5 starts here.**

POINT Statement

The POINT statement is used to establish a starting position for sequential processing of a keyed file. This statement is for use on INDEXED and RELATIVE files. CA-Easytrieve does not require that you specify the length or location of the record key field.

Data becomes available to your program only after the next successful sequential retrieval either by automatic file input or a GET statement.

Syntax

```
                  ⎧ =  ⎫ ⎧              ⎫
POINT file-name   ⎪ EQ ⎪ ⎪ field-name    ⎪  [STATUS]
                  ⎨ GE ⎬ ⎨              ⎬
                  ⎪ GQ ⎪ ⎪ literal      ⎪
                  ⎩ >= ⎭ ⎩              ⎭
```

Parameters

file-name An INDEXED or RELATIVE file described on a FILE statement in the library section of your program.

field-name or ***literal*** Any valid *field-name* or *literal* can be used as a key search value for the POINT statement. This search value is compared to the record key value in the file to determine the starting location for sequential access.

STATUS Causes the system-defined field FILE-STATUS to be set with a return code. By checking FILE-STATUS at some point in your program after coding STATUS, you can determine if the input/output request was performed properly. The FILE-STATUS field normally contains a value of zero after a successful I/O request. This parameter is also used on the GET, PUT, READ and WRITE statements.

POINT Example

The following example causes sequential processing of an INDEXED file to begin on a record with a key value of 01963 or, if no such key exists, on a record with the next higher key value.

```
FILE PERSNL INDEXED
  EMP#     9  5  N
  EMPNAME 17  8  A
  DEPT    98  3  N
  GROSS 94  4  P  2
JOB INPUT NULL NAME MYPROG
  POINT PERSNL GE '01963' STATUS
  IF FILE-STATUS NE 0 OR EOF PERSNL
    DISPLAY 'BAD POINT...FILE STATUS= ' FILE-STATUS
    STOP
  END-IF
  GET PERSNL STATUS
  IF FILE-STATUS NE 0
    DISPLAY 'BAD GET...FILE STATUS= ' FILE-STATUS
  ELSE
    DISPLAY PERSNL
  END-IF
  STOP
```

Random Access Processing

READ Statement

The READ statement provides random access to INDEXED and RELATIVE files.

Syntax

```
                      ⎧key-field-name⎫
READ file-name KEY    ⎨              ⎬  [STATUS]
                      ⎩'key-literal' ⎭
```

Parameters

file-name The *file-name* located on a FILE statement in the library section of your program.

```
⎧key-field-name⎫
⎨              ⎬  [STATUS]
⎩'key-literal' ⎭
```

Key-field-name contains the value of the record key to be found. This key value can also be expressed as a literal for INDEXED files.

READ Example

The following example involves the use of two files, PAYROLL and MASTER. PAYROLL is a sequential transaction file containing key values. MASTER is a master file which is keyed for random access.

The PAYROLL file is made available to the program through automatic input. Key values from this file, located in the EMP-NO field, are used to READ the MASTER file. READs returning non-zero FILE-STATUS values cause the DISPLAY of an error message.

```
FILE PAYROLL
  EMP-NO    1   3   N
FILE MASTER INDEXED
  EMP-NAME   40   10   A
*
JOB INPUT PAYROLL NAME READ-EXAMPLE
  READ MASTER KEY EMP-NO STATUS
  IF MASTER:FILE-STATUS NOT ZERO
    DISPLAY 'ERROR READING VSAM  +
            FILE WITH KEY: '  EMP-NO      +
             '   FILE-STATUS IS ' MASTER:FILE-STATUS
    GOTO JOB
  END-IF

    ** Logic **
```

WRITE Statement

Use the WRITE statement to add a new record, update an existing record, or delete a record from an INDEXED or RELATIVE file.

- When you use WRITE you must specify the UPDATE parameter on the FILE statement of the file being written to.
- Before you can issue a WRITE to delete or update, you must already have read the record you are writing.

WRITE Format 1

Use Format 1 when adding to or updating a record.

Syntax

```
WRITE output-file-name [UPDATE|ADD] [FROM input-file-name] [STATUS]
```

WRITE Example 1

This example involves two files, TRANS and PAYVS. TRANS is a sequential transaction file containing transaction records with a key value located in the EMP-NO field. PAYVS is a master file which is keyed for random access.

The TRANS file is made available to the program through automatic input. The EMP-NO field of the TRANS file is used as a key to READ the PAYVS file.

The value returned to the FILE-STATUS field after a READ, is checked to find out if a record with a matching key value was found. If no record was found, then an ADD is performed.

If a record was found, then an UPDATE is performed. In either case, procedures (not shown) are PERFORMed to check the FILE-STATUS value for each WRITE statement executed.

```
FILE TRANS
  EMP-NO  1  3   N
  GROSS  15  4   P  2
*
FILE PAYVS  INDEXED (UPDATE)
  GROSS  15   4  P  2
*
JOB INPUT TRANS NAME UPDATE-PGM
  READ PAYVS KEY EMP-NO STATUS
  IF PAYVS:FILE-STATUS NE 0  . * RECORD NOT FOUND
    WRITE PAYVS ADD FROM TRANS STATUS
    PERFORM ADD-STATUS-CHK
    GOTO JOB
  END-IF
  IF PAYVS:FILE-STATUS = 0  . * RECORD FOUND
    MOVE LIKE TRANS TO PAYVS
    WRITE PAYVS UPDATE STATUS
    PERFORM UPDATE-STATUS-CHK
    GOTO JOB
  END-IF
```

WRITE Format 2

Use Format 2 for deleting a record.

Syntax

```
WRITE  output-file-name  DELETE   [STATUS]
```

WRITE Example 1

```
FILE TRANS
  TRANS-KEY   14  3   A
  TRANS-CODE  17  1   A. * TRANS-CODE value of D means Delete
*
FILE PAYVS INDEXED (UPDATE)
JOB INPUT TRANS NAME VSAM-DELETE
  IF TRANS-CODE = 'D'
    READ PAYVS KEY TRANS-KEY STATUS
      IF FILE-STATUS = 0
        WRITE PAYVS DELETE STATUS
        PERFORM WRITE-STAT-CHECK
      ELSE
        DISPLAY 'ERROR IN STATUS CHECK'
      END-IF
  END-IF
```

⇒ **Reading 3 of Chapter 5 ends here.**

- **Please continue with the description of Format Determination Parameters, which is Reading 3 of Chapter 6.**

Chapter

6

Activity Section - Reporting

Introduction

One of the most powerful features of CA-Easytrieve is the easy-to-use reporting facility. Seven basic statements control most aspects of report production, including REPORT, SEQUENCE, CONTROL, SUM, TITLE, HEADING, and LINE. Special-named procedures are also available for customizing reports, allowing for great flexibility and user control. In this chapter, you'll find:

⇒ **Reading 1**

- Standard Reports
- Defining basic report characteristics with the REPORT Statement
- Spacing control parameters of the REPORT statement
- Defining report content with report definition statements including SEQUENCE, CONTROL, SUM, TITLE, HEADING, and LINE

⇒ **Reading 2**

- Label reports
- Testing aid and format determination parameters of the REPORT statement

⇒ **Reading 3**

- More format determination parameters of the REPORT statement
- Multiple Reports
- File directing parameters of the REPORT statement
- Report Procedures

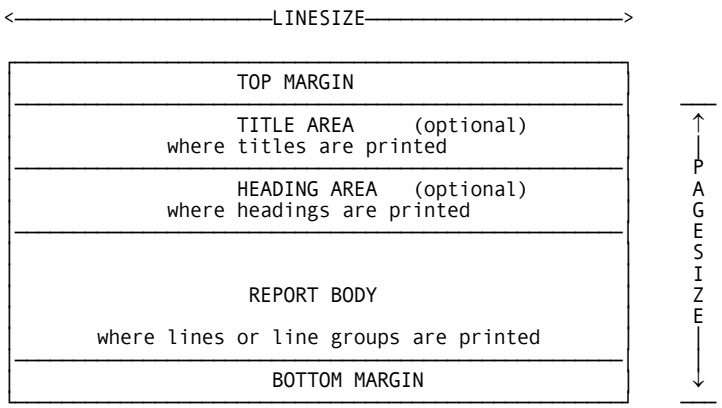
⇒ **Reading 1 of Chapter 6 starts here.**

Standard Reports

CA-Easytrieve report facility includes all of the functions necessary to produce most reports very easily. Using CA-Easytrieve report options, you can produce almost any report format. Most reports, however, are variations of what is termed the standard report. Standard reports typically consist of the following items:

- Titles
- Headings
- Lines or line groups.

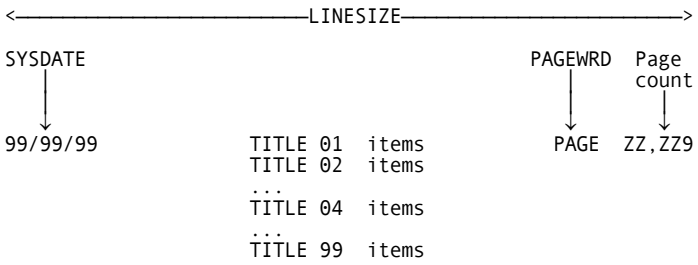
The following exhibit shows the structure of a standard report.



The following discussion tells how titles, headings, and lines are formatted and generated by CA-Easytrieve.

Titles

The title is the first item printed on each report page. The report title is specified in your program by the TITLE statement. You can have up to 99 TITLE statements in your program. The following exhibit shows the title area of a report.



When more than one TITLE statement is coded in your program, TITLE must be followed by sequence numbers (01 through 99). The following list highlights some points to remember about standard report titles:

- TITLE 01 items are printed at top-of-form.
- The current date and page count are automatically 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 sequence number controls the vertical spacing of titles relative to the first title.
- The SPACE parameter of the REPORT statement controls the number of blank characters (spaces) between title items.

The following shows two TITLE statements and resulting titles:

Statements:

```
FILE PERSNL FB(150 1800)
  DEPT  W  3  N  VALUE '903'
JOB INPUT PERSNL NAME MYPROG
  PRINT REPORT1
*
REPORT REPORT1 LINESIZE 50
  TITLE 01 'TEMPORARY EMPLOYEES'
  TITLE 03 'IN DEPARTMENT' DEPT
  LINE  01 ' '
```

Produce:

```
01/09/89       TEMPORARY EMPLOYEES    PAGE       1

               IN DEPARTMENT   903
```

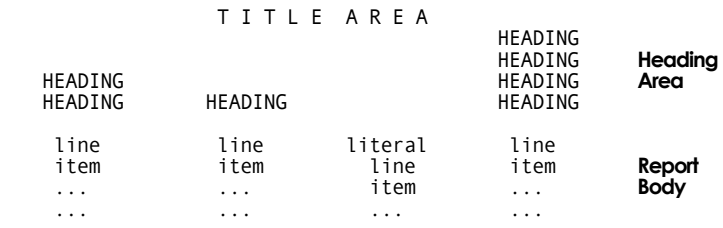
Note: A blank line was inserted where a TITLE 02 item would have otherwise been printed.

Headings

Headings in a report describe the content of line items. Line items are the single pieces of information that make up a line on a report. Usually, they form vertical columns. Each heading is centered over its associated line item. The following list highlights points to remember about headings:

- If no headings are defined, CA-Easytrieve uses the field names of the DEFINE statement as headings.
- Headings can be specified by HEADING statements in the report subactivity or by HEADING parameters on DEFINE statements.
- HEADING statements override any HEADING parameters defined for the same field.
- Line items which are literals (do not come from defined fields) do not have headings.
- Headings can be stacked (take up more than one vertical space).

The following exhibit shows the positioning of headings in a typical report:



The following shows how headings can be defined in your program.

Statements:

```
FILE PERSNL FB(150 1800)
  SSN      4  5  P  HEADING('SOCIAL' 'SECURITY' 'NUMBER')
  EMPNAME 17  20 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
```

Produce:

```
                    SOCIAL
                   SECURITY                          NET
     EMPNAME         NUMBER                          PAY

   WIMN    GLORIA    025-30-5228   * NO OVERTIME *    251.65
   BERG    NANCY     121-16-6413   * NO OVERTIME *    547.88
```

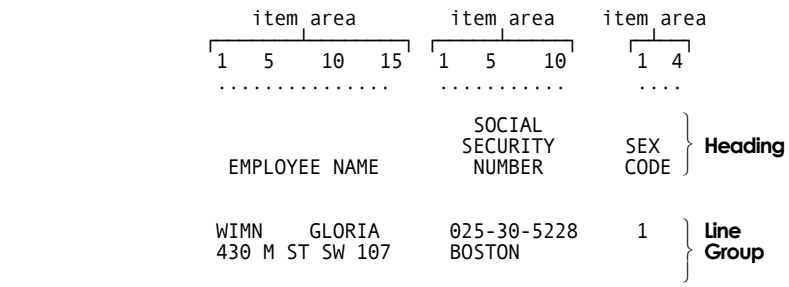
Line Group

A line is the result of a single LINE statement in your program. Each time a PRINT statement is executed, all of the fields indicated on the LINE statement are sent to the printer as a single formatted line. If there is more than one LINE statement in your program, then they 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 01 ...)
LINE 02 ...}   line group   (logical report line)
LINE 03 ...)
 ...
```

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'
  ADDR-STREET   37 20 A    HEADING 'STREET'
  ADDR-CITY     57 12 A    HEADING 'CITY'
  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 ADDR-CITY
```



See the "CA-IDMS Database Processing" chapter of the *CA-Easytrieve Programmer Guide* for more information on line item positioning.

Report Processing

The PRINT statement discussed in Chapter 5, "Activity Section - Input and Output" identifies records for output to a report and initiates the execution of a report declaration. It does not directly cause the printing of the report. Printing and formatting of a report is done by the statements that make up the report declaration (sometimes called the report subactivity since report declarations are considered subactivities of the JOB activity). There are two parts to every report declaration:

- The REPORT statement specifies the type and physical characteristics of the report.
- Report definition statements define the content of the report.

REPORT Statement

The REPORT statement is the first statement coded in a report declaration. The report statement includes the keyword REPORT and various report parameters.

Report parameters are keywords that permit you to assign values that alter the physical characteristics of the final report. Although you can specify a large number of report parameters, you can produce most reports using default (CA-Easytrieve defined) parameter values.

Report statement parameters provide you with a simple way to define tailored reports. They can be divided into three categories:

- Spacing control parameters.
- Testing aid parameters.
- Format determination parameters.

In this reading, we discuss only spacing control parameters.

Spacing Control Parameters

The following REPORT statement parameters control spacing on a standard format report:

- PAGESIZE - Lines per page (default is 58).
- LINESIZE - Length of each line (default is 132).
- SKIP - Number of blank lines to be inserted between line groups (default is 0).
- SPACE - Number of blanks inserted (horizontally) between field columns and between fields and literals in title and detail lines (default is 3).
- TITLESKIP - Number of blank lines inserted after last title line and before the first heading or detail line (default is 3).
- SPREAD - Requests that the columns of data be spread evenly over the entire line, overrides the SPACE parameter (default is NOSPREAD).
- NOADJUST - Requests title lines and report be left-justified on the page. The default is to center the report on the page. SPREAD and NOADJUST are mutually exclusive.
- NODATE - Inhibits printing the system date in positions one through eight of the first title line.

- NOPAGE - Inhibits the printing of a page number.
- NOHEADING - Inhibits the printing of column headings.

Spacing control parameters are all optional. When used, they can be coded on the REPORT statement in any order. The general format for these parameters is:

```
REPORT  report-name   +

        [PAGESIZE nn]  [LINESIZE nn]  +
        [SKIP nn] [SPACE nn]  +
        [TITLESKIP nn] +

        [          ]                          Spacing
        [SPREAD    ]                          Control
        [NOSPREAD  ]     +                    Parameters
        [          ]

        [NOADJUST]     +

        [NODATE] [NOPAGE]  +
        [NOHEADING]    +
```

REPORT Statement Example

The following program shows an example of how spacing control parameters can be used on the REPORT statement:

```
FILE PERSNL FB(150 1800)
  EMP#         9   5   N
  EMPNAME     17   8   A   HEADING ('EMPLOYEE' 'NAME')
  DEPT        98   3   N
  GROSS       94   4   P  2  MASK '$$,$$9.99'
JOB INPUT PERSNL NAME FIRST-PROGRAM
  PRINT PAY-RPT
REPORT PAY-RPT PAGESIZE 12 NOPAGE NODATE LINESIZE 70 SKIP 2 +
  SPREAD TITLESKIP 4
  TITLE 01 'EXAMPLE  PROGRAM'
  LINE 01 EMPNAME EMP# DEPT GROSS
```

The program illustrated above produces the report shown on the following page (only two pages are shown):

```
                      EXAMPLE  PROGRAM


    EMPLOYEE
      NAME           EMP#          DEPT          GROSS

    WIMN             12267         903            $373.60
```

```
                      EXAMPLE  PROGRAM


    EMPLOYEE
      NAME           EMP#          DEPT          GROSS

    BERG             11473         943            $759.20
```

Report Definition Statements

The second part of a report declaration is the report definition statements. These statements define the content of your report. When you use report definition statements, you must code them immediately after the REPORT statement, in the following order:

- SEQUENCE
- CONTROL
- SUM
- TITLE
- HEADING
- LINE

SEQUENCE Statement

The SEQUENCE statement enables you to specify the order of the lines in a report. For example, you might want reports to be in alphabetical order by name or in numerical order by employee number.

- You can sequence on any field from any input file or any W working storage field.
- You can sequence on as many fields as your system sort permits.
- Sequence fields are stated in major to minor order.
- The default sequence order is ascending. Coding D after a *field-name* reverses the order for that field only.

Syntax

```
SEQUENCE field-name [D] ...
```

SEQUENCE Examples

```
SEQUENCE CO DIV DEPT GROSS-PAY D
SEQUENCE GROUP AMT D CODE
```

CONTROL Statement

A CONTROL statement specifies that a report should automatically accumulate and print totals. A control break occurs whenever the value of any control field changes or end-of-report is reached. Control fields can be any non-quantitative field from any input file or any W working storage field. At each control break, totals are printed for any quantitative fields specified in the report.

- You can specify an unlimited number of control fields.
- Fields are coded on the CONTROL statement in major to minor order.

Syntax

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

- Final totals are automatically provided. You can alter the default by coding FINAL NOPRINT.
- NOPRINT following any field-name suppresses the printing of totals for that field (which are still accumulated) at the corresponding control break.
- NEWPAGE following any field or FINAL causes a new page after the printing of the control break totals (or, in the case of FINAL, before the printing of the final totals). Page numbers continue.
- RENUM following any field or FINAL causes a page break and restarts page numbers at 1 after the printing of the control break totals (or, in the case of FINAL, before the printing of the final totals).

Control Examples

```
CONTROL COMPANY RENUM DIV DEPT NOPRINT
CONTROL FINAL NOPRINT COMPANY NEWPAGE DIV
```

SUM Statement

The SUM statement specifies that only certain quantitative fields are to be totaled for a control report. Normally on control reports, CA-Easytrieve totals all quantitative fields specified on the LINE statement (to be discussed later). The SUM statement overrides this process; only the fields specified on the SUM statement are totaled.

- You can use SUM only in control reports.
- You can SUM any quantitative field from any active file or any **W** field.

Syntax

```
SUM field-name  ...
```

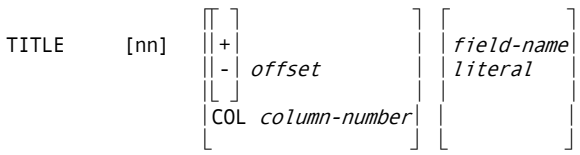
SUM Example

```
SUM GROSS NET
```

TITLE Statement

The TITLE statement allows you to define a title for your report. Up to 99 titles are permitted. You can specify literals and/or field names on the TITLE statement.

Syntax



- You use ± *offset* to alter the normal horizontal spacing between literals or fields on the title lines. Spaces are added to or subtracted from the SPACE parameter (which normally has a default of 3).
- COL *column-number* specifies the print column number where the next title item is to begin.
- If no TITLEs are coded, the date and page number, which are normally automatically included in the title, are not printed.

TITLE Examples

```
TITLE 01 'REPORT ONE'
TITLE 03 'THIS PAGE FOR DIV' -2 DIV-NO
TITLE 04 'ABC COMPANY'
```

prints:

```
01/31/91          REPORT ONE            PAGE 1

              THIS PAGE FOR DIV 15
                 ABC COMPANY
```

Control Field Values in Titles

Occasionally, you may want to print control field values in report titles. You can accomplish this by using the NEWPAGE parameter of the CONTROL statement and including a control field name on the TITLE statement. Then, control breaks occur on a new page with the control field value in the title. The following exhibit gives an example of this. (Output has been edited for the purpose of illustration.)

Statements:

```
FILE PERSNL FB(150 1800)
  EMPNAME  17   8   A
  STATE    69   2   A
  ZIP      71   5   N
  PAY-NET  90   4   P  2  MASK (A '$$,$$9.99')
JOB INPUT PERSNL NAME MY-PROG
  PRINT REPORT-1
REPORT REPORT-1 LINESIZE 65
  SEQUENCE STATE ZIP NAME
  CONTROL STATE NEWPAGE
  TITLE 01 'REPORT FOR THE STATE OF' STATE
  LINE 01 EMPNAME STATE ZIP PAY-NET
```

Produce:

```
1/17/89          REPORT FOR THE STATE OF   DC      PAGE      1

          EMPNAME    STATE    ZIP       PAY-NET

          JUDAR       DC     00000         $459.57
          PHILPS             00000         $213.76
          CROCI              20002         $215.95
          WARD               20002         $141.47
                      DC                 $1,030.75

1/17/89          REPORT FOR THE STATE OF   MD      PAGE      2

          EMPNAME    STATE    ZIP       PAY-NET

          MILLER      MD     20014         $222.61
          PETRIK             20014         $154.70
          LACH               20028         $215.91
          VETTER             20028         $189.06
                      MD                   $782.28

1/17/89          REPORT FOR THE STATE OF   VA      PAGE      3

          EMPNAME    STATE    ZIP       PAY-NET

          MCMAHON     VA     22202         $283.19
          CORNING            22204         $103.43
          BYER               22207         $259.80
          ARNOLD             22209         $356.87
                      VA                   $939.03

                                         $2,752.06
```

HEADING Statement

As with the DEFINE statement in the library section, you can define an alternate column heading for a field in the report declaration. The HEADING statement overrides a HEADING parameter coded for the same field in the library section of the program. When alternate headings are not defined, either by a HEADING statement or the HEADING parameter of DEFINE, then the field name is used as the heading.

- Use one HEADING statement per field.
- Words in a heading can be stacked to save space in the column. This is done by placing individual words in single quotes.

Syntax

```
HEADING  field-name ('heading-literal'...)
```

HEADING Example 1

```
HEADING EMP-NO 'EMP NO'
```

prints a column heading on the report that looks like:

```
EMP NO
```

HEADING Example 2

```
HEADING SSN ('SOCIAL' 'SECURITY' 'NUMBER')
```

prints a stacked column heading:

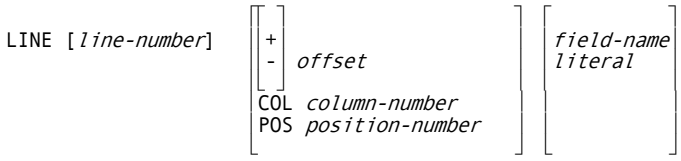
```
 SOCIAL
SECURITY
 NUMBER
```

LINE Statement

The LINE statement defines the content of a report line. Multiple LINE statements define a line group. Use LINE 01 to designate headings for the report columns.

- You can specify up to 99 lines per record.
- You can specify any field from an input file or working storage.

Syntax



- When literals are specified they print on all lines but are not used as headings.
- ± *offset* is used to alter the normal spacing between line items. *nn* is added to or subtracted from the SPACE parameter (which normally has a default of 3).
- COL (column) specifies the print column number where the next field is to begin.
- POS (position) provides for aligning fields under the corresponding column heading positions indicated on the LINE 01 statement.

LINE Example

```
LINE 01 DEPT DIV EMPNAME
LINE 02 POS 2 CODE POS 3 ADDRESS
LINE 03 POS 3 CITY-STATE
```

prints the field's contents in the following format:

```
DEPT          DIV       EMPNAME

911           02        MATT JONES
              512       2232 HILL
                        ANYWHERE IL
```

⇒ **Reading 1 of Chapter 6 ends here.**

- **If you have completed the tutorial in Chapter 2, go on to the beginning of Chapter 7.**
- **If you branched to this chapter from the tutorial in Chapter 2, go back to Lesson 5 in Chapter 2.**

⇒ **Reading 2 of Chapter 6 starts here.**

Label Reports

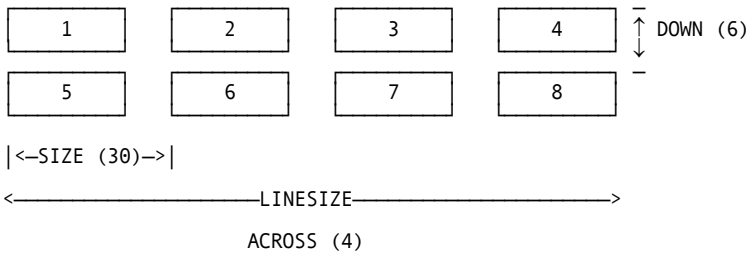
You can use the label report capability of CA-Easytrieve to print mailing labels and other applications that require inserting variable data in a repetitious format. A label report is different from a standard report in the following ways:

- Label reports do not have titles and headings.
- Multiple labels can be printed side-by-side.
- Controlled label reports permit 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 (discussed later in this chapter).

You can use the label report function whenever a complete logical print page is to be produced by each PRINT statement.

Label Format

Label reports are specified by use of the LABELS option of the REPORT statement. 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 previous exhibit, labels 1 through 4 compose a label line. A single line group composes each label. Therefore, CA-Easytrieve produces a label for each PRINT statement execution. CA-Easytrieve formats the labels on the page in the order shown in the above exhibit. DOWN and SIZE (subparameters of the LABELS option) indicate the dimensions of each label.

Format Determination Parameters

Format determination parameters are parameters of the REPORT statement which determine the type of report to be printed. The LABELS parameter is responsible for formatting reports which print mailing labels.

LABELS specifies that the report will be in label format rather than the standard report format. It automatically inhibits the printing of the date, page, headings, and titles. The following subparameters are used with LABELS.

- ACROSS specifies the number of labels printed across the print line (default is 4).
- DOWN specifies the number of lines down from the first line of the first label to the first line of the second label (default is 6).
- SIZE specifies the number of print positions from the first position on the first label to the first position on the second label (default is 30).

The LABELS parameter has the following format:

```
[LABELS ]      +
  ([ACROSS nn]
   [DOWN nn]
   [SIZE nn])
```

Format Determination Parameters

REPORT Statement Example

The following program creates a label report (shown on the next page):

```
FILE PERSNL FB (150 1800)
  NAME       17   8   A
  ADDRESS    37  39   A
    ADDR-STREET 37  20  A
    ADDR-CITY   57  12  A
    ADDR-STATE  69   2  A
    ADDR-ZIP    71   5  N

JOB INPUT PERSNL NAME FIRST-PROGRAM
  PRINT PAY-RPT

REPORT PAY-RPT LABELS (ACROSS 3 SIZE 23)
  LINE 01 NAME
  LINE 02 ADDR-STREET
  LINE 03 ADDR-CITY ADDR-STATE
  LINE 04 ADDR-ZIP
```

The following report is created by the previous program:

```
WIMN                 BERG                 CORNING
430 M ST SW 107      3710 JENIFER ST N W  3208 S 5TH
WASHINGTON     DC    WASHINGTON     DC    ARLINGTON      VA
20004                20015                22204


NAGLE                ARNOLD               MANHART
826 D STREET SE      1569 COLONIAL TERR A 1305 POTOMAC ST N W
WASHINGTON     DC    ARLINGTON      VA    WASHINGTON     DC
20003                22209                20007


TALL                 BRANDOW              LARSON
1412 36TH ST NW      3616 B ST S E        610 H ST SW
WASHINGTON     DC    WASHINGTON     DC    WASH           DC
20007                20019                20024


BYER                 HUSS                 POWELL
3400 NORTH 18TH STRE 1355 TEWKESBURY PLAC 5023 AMES STREET N E
ARLINGTON      VA    WASHINGTON     DC    WASHINGTON     DC
22207                20012                20019
```

Testing Aid Parameters

Testing aid parameters are provided as a testing aid for report development. You can run your newly developed report programs against real data while limiting the amount of information printed. There are two testing aid parameters of the REPORT statement, LIMIT and EVERY.

Testing aid parameters have the following format:

```
REPORT   [report-name]  +
  [LIMIT number-of-records]
  [EVERY n-number-of-lines]
```

- LIMIT option limits the number of records processed by the report. The value, *number-of-records*, can be any integer literal in the range 1 through 32,767.
- EVERY option enables you to specify that only every Nth line is printed in the report. The value of *n-number-of-lines* can be any integer literal in the range 1 through 32,767.

⇒ **Reading 2 of Chapter 6 ends here.**

- **Please continue with the description of Formatting a Screen Item for Display, which is Reading 2 of Chapter 7.**

⇒ **Reading 3 of Chapter 6 starts here.**

Format Determination Parameters

Aside from the format determination parameter LABELS used (on the REPORT statement) to format labels reports, there are six other format-related parameters of the REPORT statement. We'll discuss four of them here. Each parameter and its purpose is given in the following table:

Parameter	Purpose
DTLCTL	(Detail Control) is used to control the printing of control fields on detail lines of a control report.
SUMCTL	(Sum Control) is used to control the printing of control fields on total lines of a control report.
SUMMARY	Inhibits the printing of detail data on control reports. Permits only the printing of total lines.
SUMFILE	Generates a file containing control fields and totals during generation of a control report.

The format of these four parameters is:

```
REPORT [report-name] +

[SUMMARY] +
[SUMFILE summary-file-name] +

[      {EVERY} ]
|DTLCTL{FIRST} | +
[      {NONE } ]

[       {   [ALL ]                 } ]
|SUMCTL { ( |HIAR|[DTLCOPY   ] )  } | +
|       {   |NONE|[DTLCOPYALL]    } | |
[       {   [TAG ]                 } ]
```

A discussion of DTLCTL, SUMCTL, SUMMARY, and SUMFILE follows.

DTLCTL Parameter

The DTLCTL (Detail Control) parameter of REPORT establishes the method for printing control field values on detail lines of a control report by using the subparameters EVERY, FIRST, and NONE. The following exhibit shows an example program using DTLCTL options. The program shown can be run with any of the three options. See the "Report Processing" chapter in the *CA-Easytrieve Programmer Guide* for more information.

```
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 (* replace with one: EVERY, FIRST, or NONE *)
  SEQUENCE STATE ZIP LAST-NAME
  CONTROL  STATE ZIP
  LINE 01  LAST-NAME STATE ZIP PAY-NET
```

SUMCTL Parameter

The SUMCTL (Sum Control) parameter of REPORT 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 field values on total lines and is shown along with the SUMMARY parameter, later in this chapter.) The following exhibit shows an example program using these options. The program shown can be run with any of the four options. See the "Report Processing" chapter of the *CA-Easytrieve Programmer Guide* for more information.

```
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
    (* replace with one: ALL, HIAR, NONE, or TAG *)
  SEQUENCE STATE ZIP LAST-NAME
  CONTROL  STATE ZIP
  LINE 01  LAST-NAME STATE ZIP PAY-NET
```

SUMMARY Reports

SUMMARY is a parameter of the REPORT statement that causes the report to print as a summary report.

Summary reports consist of only total lines which normally include only control fields and totals. All detail lines are inhibited from printing.

DTLCOPY Subparameter

It can be helpful on summary reports to have detail field information printed on the total lines to provide greater readability. The DTLCOPY option of the SUMCTL parameter of the REPORT statement, copies detail fields (non-control and non-total fields) as they appear just before the control break, onto the total lines of the summary report.

The exhibit that follows shows a program that produces a summary report and includes the DTLCOPY option. If this option was not used, the LAST-NAME values would not print.

DTLCOPY causes the detail information to be printed only on the first control level of the report. DTLCOPYALL prints the detail to be printed on all summary 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 DTLCOPY
SEQUENCE STATE ZIP LAST-NAME
  CONTROL  STATE ZIP
  LINE 01  LAST-NAME STATE ZIP PAY-NET
```

Data:

```
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
```

Produce:

```
Line
Description     LAST-NAME   STATE    ZIP     PAY-NET

ZIP total         BROWN      IL     60076      802.35
ZIP total         JONES      IL     60077      641.97
STATE total                  IL                1444.32
ZIP total         SMITH      TX     75218      777.77
STATE total                  TX                777.77

FINAL total                                    2222.09
```

Summary Files

A summary file, containing 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 through the standard report facilities of CA-Easytrieve. You can request the summary file by defining the file in the library and then referencing it with the REPORT SUMFILE parameter. See the "Report Processing" chapter of the *CA-Easytrieve Programmer Guide* for more information.

Multiple Reports

Multiple Reports to a Single Printer

Several reports can be produced simultaneously with one pass of the input file. No special coding is needed for multiple reports on the same printer. The following program produces three reports from one input file:

```
FILE PERSNL FB(150 1800)
  EMPNAME       17   8   A
  DEPARTMENT    98   3   N
  NET           90   4   P 2
  GROSS         94   4   P 2
  DEDUCTIONS     W   4   P 2
```

Continued

Continued

```
JOB INPUT PERSNL NAME MULTRPTS
  PRINT RPT1
  DEDUCTIONS = GROSS - NET
  PRINT RPT2
  IF DEPARTMENT = 911
    PRINT RPT3
  END-IF
*
REPORT RPT1
  TITLE 1 'REPORT ONE'
  LINE 1 EMPNAME DEPARTMENT GROSS NET
*
REPORT RPT2
  SEQUENCE DEPARTMENT
  TITLE 1 'REPORT TWO'
  LINE 1 DEPARTMENT EMPNAME GROSS NET DEDUCTIONS
*
REPORT RPT3
  CONTROL
  TITLE 1 'REPORT THREE - DEPT 911'
  LINE 1 EMPNAME GROSS NET DEDUCTIONS
```

REPORT ONE (RPT1) produces a very simple listing of all employees.

REPORT TWO (RPT2) gives the same information as REPORT ONE but includes an additional column with the deductions printed.

REPORT THREE (RPT3) produces a report that contains only information from department 911.

Multiple Reports to More Than One Printer

Multiple reports in one program can be sent to multiple output devices or multiple printers. This can be an effective way of economizing on processing time if your site supports multiple output devices.

FILE Directing Parameters

PRINTER Parameter

The PRINTER parameter of the REPORT statement directs the report's printed output to a file other than the system default output device (SYSPRINT/SYSLST). Such files must be defined in the library section by a FILE statement which also contains a PRINTER parameter.

The REPORT statement must identify the appropriate *file-name*, using the PRINTER parameter in the following format:

```
REPORT  report-name
  [PRINTER  file-name]
```

The following exhibit shows a CA-Easytrieve program that produces two reports, each sent to separate printers.

```
FILE PAYFILE
  EMPNAME      17      16      A
  ADDRESS      57      20      A
  STREET       37      20      A
  EMP-NUMBER    9       5      N
*
FILE SPFORM PRINTER
*
JOB INPUT PAYFILE NAME MULT-PRINTERS
  IF EMP-NUMBER LE 12345
    PRINT FIRST-REPORT
    PRINT NORM-REPORT
  END-IF
*
REPORT FIRST-REPORT PRINTER SPFORM
  SEQUENCE EMP-NUMBER
  LINE 1 EMPNAME
  LINE 3 STREET
  LINE 5 ADDRESS
*
REPORT NORM-REPORT
  LINE 1 EMPNAME ADDRESS EMP-NUMBER
```

The first report declaration produces a report to a print output file designated SPFORM in the second file statement. This print file gets tied to a physical printer through your Job Control Language (JCL) statements.

The second report declaration produces a report that is output to the printer you normally use with other CA-Easytrieve programs (the default system output device).

Report Procedures (PROCs)

Report Procedures (PROCs) are user-defined routines that are automatically invoked within a report declaration to perform special data manipulation not included in the logic subactivity. There are seven report PROCs in CA-Easytrieve.

Code any report procedures immediately after the last LINE statement of each report in your program. Report procedures are identified in your program by the" . PROC" keyword, as shown below:

```
  REPORT statement
   LINE statement
┌                    ┐
| REPORT-INPUT. PROC |
| BEFORE-BREAK. PROC |
| AFTER-BREAK. PROC  |
| BEFORE-LINE. PROC  |
| AFTER-LINE. PROC   |
| ENDPAGE. PROC      |
| TERMINATION. PROC  |
└                    ┘
   ** procedure logic **
  END-PROC
```

- You must code an END-PROC at the end of each procedure.
- You code the logic to be executed in a report PROC the same way you code logic in a JOB activity.
- DISPLAY is the only input or output operation permitted.
- Although you can code these *procs* in any order, each *proc* can only be used once per report.

REPORT-INPUT. PROC

The REPORT-INPUT. PROC allows for final screening and modification of report input data. It is performed for each record selected for the report that contains the PROC.

- If you code a REPORT-INPUT procedure, then you must execute a SELECT statement in the PROC to cause data to continue to the report.
- If a report has been SEQUENCEd, this procedure is invoked after each record is output from the system sort.

REPORT-INPUT Example

Statements:

```
FILE PERSNL
  BRANCH          2   2   N
  EMP#            9   5   N
  EMPNAME        17  20   A
  PAY-NET        90   4   P 2
  TOT-NET         S   5   P 2
  PCT-NET-TO-TOT  W   3   P 1
*
JOB INPUT PERSNL NAME RPTINPT
  TOT-NET = TOT-NET + PAY-NET
  PRINT PCT-RPT
*
REPORT PCT-RPT LIMIT 20
  SEQUENCE BRANCH EMP#
  CONTROL FINAL NOPRINT BRANCH NOPRINT
  TITLE 1 'EXAMPLE OF REPORT-INPUT PROC'
  LINE 1 BRANCH EMPNAME EMP# PAY-NET PCT-NET-TO-TOT
*
```

```
REPORT-INPUT. PROC
  PCT-NET-TO-TOT = PAY-NET / TOT-NET * 100 + .05
  SELECT
END-PROC
```

Produce:

```
01/31/91             EXAMPLE OF REPORT-INPUT PROC           PAGE   1

                                EMPLOYEE          NET
 BRANCH    EMPLOYEE NAME        NUMBER           PAY     PCT-NET-TO-TOT

    1    BRANDOW LYDIA          02200             554.31          4.4
         HUSS    PATTI          11376             223.71          1.8
         WIMN    GLORIA         12267             251.65          2.0

    2    NAGLE   MARY           00370             340.59          2.7
         KRUSE   MAX            03571             182.09          1.5
         BERG    NANCY          11473             547.88          4.4
         POWELL  CAROL          11710             167.96          1.3

    3    PETRIK  KATHY          00577             154.70          1.2
         CORNING GEORGE         02688             103.43           .8
         DENNING RALPH          02765             109.60           .9
         FORREST BILL           03416              13.19           .1
         MCMAHON BARBARA        04234             283.19          2.3
         MANHART VIRGINIA       11602             250.89          2.0

    4    POST    JEAN           00445             206.60          1.7
         ARNOLD  LINDA          01963             356.87          2.9
         LARSON  RODNEY         11357             215.47          1.7
         BYER    JULIE          11467             259.80          2.1
         TALL    ELAINE         11931             355.19          2.8

    5    VETTER  DENISE         01895             189.06          1.5
         LOYAL   NED            04225             230.50          1.8
```

BEFORE-BREAK. PROC

The BEFORE-BREAK. PROC allows for modification of totals and special annotation before total line printing caused by the CONTROL statement. A system-defined field named LEVEL can be tested to determine the appropriate break:

```
LEVEL = 1 for minor break
      = 2 for next break
      = N + 1 for final totals.
       (N is the number of control fields)
```

In the following example, the BEFORE-BREAK. PROC causes the DEPARTMENT annotation at each of the breaks, and modifies the total in PCT to be the percent, based on total amounts.

Statements:

```
FILE PAYROLL
  EMP#          9    5 N    HEADING ('EMPLOYEE' 'NUMBER')
  NET          90    4 P 2  HEADING ('NET' 'PAY')
  DEPT         98    3 N
  GROSS        94    4 P 2  HEADING ('GROSS' 'PAY')
  DED           W    3 P 2
  PCT           W    4 N 2
JOB INPUT PAYROLL NAME CORRECT-PCT
  IF DEPT = 911 914 921
    DED = GROSS - NET
    PCT = DED / GROSS * 100
```

```
    PRINT PCT-REPORT
  END-IF
REPORT PCT-REPORT LINESIZE 73
  SEQUENCE DEPT
  CONTROL FINAL NOPRINT DEPT NOPRINT
  TITLE 1 'THIS REPORT WILL ILLUSTRATE USE OF'
  TITLE 2 'BEFORE-BREAK PROCEDURE'
  LINE DEPT EMP# GROSS NET DED PCT
BEFORE-BREAK. PROC
  PCT = DED / GROSS * 100
  IF LEVEL = 1. * DEPT TOTALS
    DISPLAY SKIP 1 'DEPARTMENT ' DEPT POS 3 GROSS POS 4 NET +
      POS 5 DED POS 6 PCT
    DISPLAY SKIP 1
  END-IF
  IF LEVEL = 2. * FINAL TOTALS
    DISPLAY SKIP 1 'FINAL' POS 3 GROSS POS 4 NET +
      POS 5 DED POS 6 PCT
  END-IF
END-PROC
```

Produce:

```
01/31/91          THIS REPORT WILL ILLUSTRATE USE OF        PAGE 1
                        BEFORE-BREAK PROCEDURE

            EMPLOYEE    GROSS           NET
DEPT         NUMBER      PAY            PAY         DED       PCT

911          00445        292.00         206.60     85.40    29.24
             11710        243.24         167.96     75.24    30.93
             11357        283.92         215.47     68.45    24.10
             01963        445.50         356.87     88.63    19.89
             09764        121.95          96.64     25.31    20.75
             04589        313.60         229.69     83.91    26.75
             05805        174.15         134.03     40.12    23.03
             03890        386.40         272.53    113.87    29.46
             12461        313.60         219.91     93.69    29.87
             12829        365.60         238.04    127.56    34.89
             01730        315.20         202.43    112.77    35.77
             03571        242.40         182.09     60.31    24.88

DEPARTMENT    911       3,497.52       2,522.26    975.26    27.88

914          07231      1,004.00         685.23    318.77    31.75
             08262        376.00         215.95    160.05    42.56
             10961        399.20         291.70    107.50    26.92
             11602        344.80         250.89     93.91    27.23
             00185        279.36         189.06     90.30    32.32

DEPARTMENT    914       2,403.36       1,632.83    770.53    32.06

921          00577        220.80         154.70     66.10    29.93
             11376        360.80         223.71    137.09    37.99
             05482        183.75         141.47     42.28    23.00

DEPARTMENT    921         765.35          519.88    245.47    32.07

FINAL                   6,666.23       4,674.97    1991.26   29.87
```

AFTER-BREAK. PROC

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. In the following exhibit, the total line for the second control field ZIP receives special annotation:

AFTER-BREAK Example

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. PROC

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.

ENDPAGE Example

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. PROC

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:

TERMINATION Example

Statements:

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

Produce:

```
...
╠═══════════════════════════════════════════════════════════════╣
  2222.09 IS THE Y-T-D COMPANY NET PAY


  PLEASE ROUTE THIS REPORT TO CORPORATE OFFICERS
```

BEFORE-LINE. PROC and AFTER-LINE. PROC

A BEFORE-LINE procedure is invoked immediately before, and an AFTER-LINE procedure immediately following, the printing of each detail line.

A BEFORE-LINE/AFTER-LINE procedure is commonly used to print an annotation before/after a detail line on the report. 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
```

Produces:

```
               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
```

⇒ **Reading 3 of Chapter 6 ends here.**

- **Please continue with the description of Determining the Cursor Location, which is Reading 3 of Chapter 7.**

Chapter

7

Activity Section - Screens

Introduction

CA-Easytrieve allows you to display and receive information from an online terminal. Five basic statements control most aspects of screen processing: SCREEN, TITLE, ROW, KEY, and MESSAGE. Special-named procedures are also available to perform customized actions specific to your application. In this chapter, you'll find:

⇒ **Reading 1**

- Basic screen format
- SCREEN activity structure, statement, display attributes
- Screen title area, TITLE statement
- Screen work area, ROW statement, edit masks
- Screen message area, MESSAGE statement
- Screen function key area, KEY statement
- Special-named screen procedures

⇒ **Reading 2**

- Formatting a screen item for display: JUSTIFY and FILL
- Automatic editing of input: UPPERCASE and VALUE
- Cursor positioning on a screen
- KEY statement: branch actions and IMMEDIATE processing
- Screen procedures: branch actions

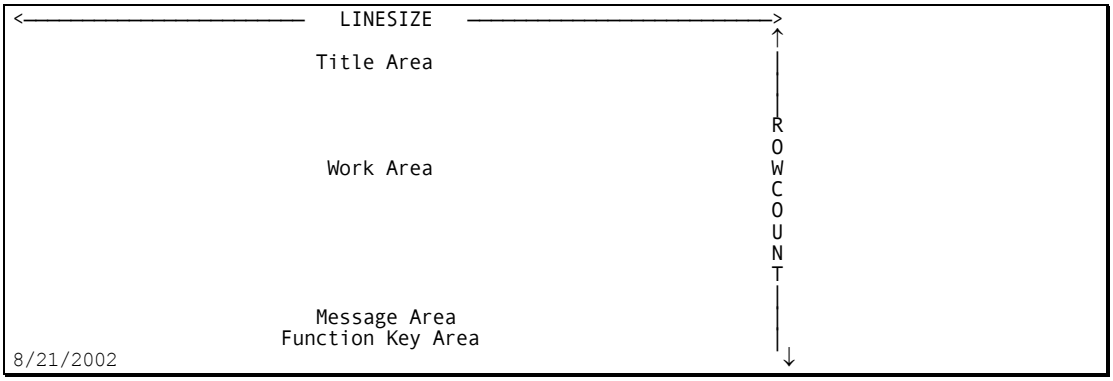
⇒ **Reading 3**

- Determining the cursor location
- Testing for field modification
- Overriding system-defined attributes and message locations: DEFAULT statement
- Overriding standard screen sizes

⇒ **Reading 1 of Chapter 7 starts here.**

Basic Screen Format

CA-Easytrieve screen format is illustrated below. The size of the screen defaults to the values specified by your system administrator when CA-Easytrieve was installed.



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, are located at the top of the screen, and are automatically centered. 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.

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 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 to define the function key area.

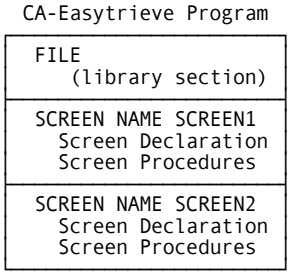
SCREEN Activity

You use a SCREEN activity to describe and process an online 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:

- The screen declaration statements (TITLE, ROW, KEY, MESSAGE) that define the contents of the screen.
- The optional special-named screen procedures that enable you to code procedural logic to perform file I/O or complex editing.

The following exhibit illustrates the basic structure of screen processing in a CA-Easytrieve program. You can define one or more screens for each program.



SCREEN Statement

You use the SCREEN statement to name and initiate a SCREEN activity. It must be the first statement in your SCREEN activity.

Syntax

```
SCREEN [NAME screen-name] [UPPERCASE]
```

- Optionally, specify a name for the SCREEN activity. The name can:
  - Be up to 128 characters in length
  - Contain any character other than a delimiter
  - Begin with A-Z, 0-9, or a national character (#, @, $)

  - Not consist of all numeric characters.
- Specify UPPERCASE to translate the data received from the terminal to upper case before it is processed. If UPPERCASE is not specified, the data is processed as the user enters it.

SCREEN Activity 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 activity 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 Items

Screen items consist of the fields and literals that you want to display to or receive from the terminal user. Unless you code 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 by using the COL parameter of the TITLE or ROW statement. See the TITLE or ROW Statement, later in this chapter, for more information.

You must ensure that fields used on a screen are in available storage. This requires that you code either WORKAREA on FILE statements for fields used on the screen and explicitly initialize the fields or that you execute an input statement to fill the fields with data prior to displaying the screen. See the *CA-Easytrieve Language Reference Guide* for the complete syntax of the FILE statement.

Screen Item Attributes

The space preceding each screen item contains system information describing the **screen attributes** for the item. Screen 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 specify screen attributes for each individual item on the screen by coding one or more attribute keywords on the ATTR parameter of the TITLE or ROW statement. If you do not specify attributes for each item, CA-Easytrieve uses the default attributes specified by your system administrator when CA-Easytrieve was installed.

Note: You can also override attributes at a screen level. See "Overriding System-Defined Attributes and Message Locations," later in this chapter.

Valid attributes are:

- **SENDONLY** - specifies that the field is not to be received (it is ignored if entered).
- **CURSOR** - place the cursor on this field when displayed on the terminal. If more than one field contains the CURSOR attribute, the cursor is placed on the first field that contains CURSOR.
- **ASKIP** or **PROTECT** - ASKIP specifies that the field is an auto-skip field. PROTECT specifies that the field is protected and not auto-skipped. If neither is specified, the field is unprotected.
- **NUMERIC** - specifies that only numeric data can be entered in this screen field.
- **INTENSE** or **INVISIBLE** - INTENSE specifies that the field displays brightly. INVISIBLE specifies that the field is present on the screen but is not displayed.

- **GREEN**, **RED**, **BLUE**, **TURQ** or **TURQUOISE**, **PINK**, **YELLOW**, **WHITE** - The value specified is the color of the field or literal when displayed on a screen.
- **MUSTFILL** - requires that all spaces have a non-blank character typed into them.
- **MUSTENTER** - sends an error message to the terminal if the field was not changed.
- **TRIGGER** - causes the screen to be received as soon as the terminal operator has modified the field and tries to move the cursor out of the field.
- **BLINK**, **REVERSE**, or **UNDERLINE** - BLINK displays the item blinking. REVERSE displays the item in reverse video. UNDERLINE displays the item underlined.
- **ALARM** - causes the terminal alarm to sound.

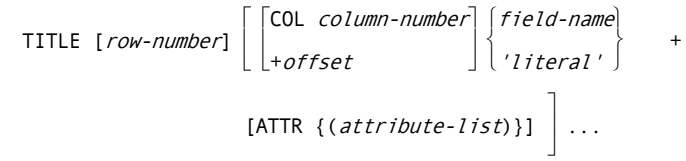
Screen Title Area

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 a TITLE statement coded after the SCREEN statement in the SCREEN activity.

TITLE Statement

The TITLE statement is used to automatically center items for display on a screen.

Syntax



- Specify the *row-number* on which you want the TITLE to be displayed. If *row-number* is not specified, the next screen row is used for the title. The next screen row is not the highest row used, but the previously-specified row plus one. If no rows are previously specified, row one is used.
- Use COL to display a title item at a specific column (*column-number*) on the screen.
- Title items are separated by one space on a screen. Use +*offset* to add additional spaces between title items. Keep in mind that a syntax error occurs when a TITLE item overlays another screen item.

- Specify one or more *field-name*s or *'literal*s' for the title. *Field-name* is the name of a field to be displayed as a title on the screen. *'Literal'* is an alphanumeric string to be displayed as a title on the screen.
- Specify a list of attribute keywords for the title item. Remember, fields in a title are always for display only. If you do not specify attributes for each title item, CA-Easytrieve uses the default attributes specified by your system administrator when CA-Easytrieve was installed.

Note: You can also override attributes at a screen level. See "Overriding System-Defined Attributes and Message Locations," later in this chapter.

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 activity. Each ROW statement describes the fields and literals to be located on each row of the screen.

ROW Statement

The ROW statement specifies the items (fields or literals) to be displayed or received on a row of a screen. Multiple items can be coded on each ROW statement. Attributes can be specified for each literal coded on the ROW statement. Attributes and edit masks can be specified for each field-name coded on the ROW statement.

Syntax

```
ROW [row-number] +

[+offset-value    ] [field-name   ] +
[COL column-number] ['row-literal']

[ATTR {(attribute-list)} ] +

[MASK ({[mask-identifier] [BWZ] ['mask-literal'] | HEX})]
[NOMASK                                                ]  ...
```

- *Row-number* specifies the line on which the item on the screen is displayed. A ROW without a *row-number* is assigned the next row number on the screen. Next is defined as the previous *row-number* plus one, not the highest number used as yet.
- A ROW without any fields or literals displays a blank line on the screen at the corresponding row-number.
- The +*offset-value* or the COL *column-number* parameter permits you to control positioning of an item on the row.
- +*Offset-value* is the number of columns (spaces) preceding a screen item. The default +*offset-value* is +1 because the space preceding each screen item is reserved for screen attributes.
- Use *column-number* to explicitly specify the column at which the screen item is displayed.
- If you do not code an +*offset-value* or *column-number*, the next *field-name* or *'row-literal*' is displayed one column after the end of the previous *field-name* or *'row-literal*' on the same row.
- *Field-name* can be any field you defined in your program.
- *'Row-literal*' can be any text you want to display on the screen.
- ATTR specifies one or more attribute keywords for the row item. See the list of valid attributes earlier in this chapter.

- The optional MASK parameter is used to format a numeric field for display.
- If MASK is not coded, the MASK coded on the field's definition is used. Use NOMASK to specify that the field's definition MASK not be used. See "Describing Files and Fields" in Chapter 3, "Library Section - Describing and Defining Data" for more information.
- Any letter from A through Y can be used as an optional *mask-identifier*. You can use the letter to identify a new mask or to retrieve a mask that was previously defined by your system administrator or by a mask parameter on a previous field definition or ROW usage. If the new mask that you identify does not already exist, CA-Easytrieve retains the mask for future reference. Do not use the same identifier to establish more than one mask.
- The BWZ (blank when zero option suppresses the display of *field-name* when it contains all zeros. BWZ can be used by itself or with other options on the MASK parameter.)
- '*Mask-literal*' defines an edit mask and must be enclosed within single quotes. The actual edit mask is coded according to the rules specified under the "MASK Parameter" in Chapter 3, "Library Section - Describing and Defining Data."
- Specify HEX to display the field in double-digit hexadecimal format. You can display fields of up to 50 bytes with the HEX mask. HEX edit masks are not permitted for VARYING fields.
- When fields are received from the terminal, the mask is used as an editing template, also. Special characters in the MASK are stripped from the data before it is moved into the field data area. See the *CA-Easytrieve Programmer Guide* for more information.

Location Example

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 Example

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.

```
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
```

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'
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
```

```
                                   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
```

Hexadecimal Mask Example

CA-Easytrieve enables you to display data in hexadecimal format. A hexadecimal mask can be applied to fields of any data type, including alphanumeric. This permits you to display the 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  . . . . C2C5D9C7404040404040404040404040404040404040
 Gross Pay . . 0075920C
```

Screen Message Area

The message area displays system and programmer-issued messages to the terminal user. CA-Easytrieve enables you to issue different levels of messages depending on the severity of the error using the MESSAGE statement. 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 are always message level ACTION.

MESSAGE Statement

The MESSAGE statement enables you to issue your own specific messages for a screen activity. You define the message type and specify the message text using the MESSAGE statement.

Syntax

```
        ┌'literal' ┐
MESSAGE ┤          ├  ...  +
        └field-name┘

    ┌      ┌INFORMATION┐ ┐
    │LEVEL ┤WARNING    ├ │
    └      └ACTION     ┘ ┘
```

- Use *literal* to define the text you want displayed in the message. Use *field-name* to specify a field whose contents you want displayed as part of the message. A message can consist of a combination of *literals* and *field-names*.

- The maximum length of a message is 130 characters. If the message exceeds the message area for the screen on which it is displayed, the message is truncated.
- Use LEVEL to specify the type of message you are defining.

Message Area Location

The default message area location is at the bottom of the screen, just above the function key display area. All three levels of messages are sent to the same screen row number. If two messages are sent at the same time, the message with the highest severity is displayed. The severity precedence from highest to lowest is:

- ACTION
- WARNING
- INFORMATION

If multiple MESSAGE statements of the same precedence are issued before displaying the screen, the last message issued is displayed.

Message Attributes

Screen attributes for the three levels of messages are set by your system administrator when CA-Easytrieve is installed.

Note: You can override system-defined message attributes and message locations with a DEFAULT statement. See "Overriding System-Defined Attributes and Message Locations," later in this chapter.

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 issue messages from special-named screen procedures that you code following the SCREEN activity by executing the MESSAGE statement prior to the display of the screen.

Example

```
MESSAGE 'Department of ' EMP-DEPT ' not 900-999.' LEVEL ACTION
```

Screen Function Key Area

The function key area is used to tell the terminal user which function keys are active and the action each performs. You use KEY statements to define the function key area on a screen.

KEY Statement

The KEY statement is used to:

- Define valid keys for the screen
- Specify descriptive text to be displayed for each valid key
- Assign automatic functions to be executed for each valid key.

Syntax

```
KEY key-name [THRU key-name]...[NAME 'literal']
```

- Each KEY statement specifies one or more terminal keys that are valid on the screen. If the user presses an invalid key, CA-Easytrieve automatically issues an error message.
- Specify a symbolic name for a terminal key as described by the system-defined field, KEY-PRESSED. KEY-PRESSED is a two-byte binary field that contains a value representing the most recent terminal key pressed by the terminal user.
- CA-Easytrieve automatically defines symbolic names that correspond to values for the most common keys.

Terminal Key	Symbolic Name	Constant Value
Enter	ENTER	1
Clear	CLEAR	11
PA1 thru PA3	PA1 thru PA3	12 thru 14
PF1 thru PF24	F1 thru F24	21 thru 44
F1 thru F12	F1 thru F12	21 thru 32

- Only terminal keys with a KEY-PRESSED symbolic name can be used on a KEY statement. If other terminal keys (for example, test request) are required, you must test KEY-PRESSED using the constant value of the terminal key in your program code. If you test for terminal keys without a symbolic name, you cannot code KEY statements in your program. (In this case, all terminal keys are considered valid for the screen.)

- Use THRU *key-name* to specify a range of *key-names*. A range of *key-names* includes all keys whose constant values for KEY-PRESSED fall between the constant values of the keys you specify for the range. For example, if you code:

  ```
  KEY CLEAR THRU F12
  ```

  the PA1, PA2, and PA3 keys are also valid. The constant values of the PA keys (12, 13, 14) fall between the value for CLEAR (11) and F12 (32).
- You can also specify a series of non-consecutive *key-names* by omitting THRU. Optionally, you can separate a series of *key-names* with commas for readability.
- You can specify a range of *key-names* and a series of *key-names* on the same KEY statement. See the examples below.
- The optional NAME parameter permits you to specify descriptive text to be displayed with the key on the screen. The format is:

  ```
  key-name=literal
  ```

  For example:

```
F1=Help  F3=Exit  F12=Cancel
```

- *'Literal'* can contain a maximum of 20 characters.
- To display only the *key-name* on a screen, code NAME *'literal'* with a blank space between single quotes (' ').
- If you do not code NAME, no display is created for the key.

Location

The function key display area is built on the bottom line of a screen. If the key display area requires additional lines because of the number of keys and the length of the descriptive text you specify, additional lines at the bottom of the screen are used. The function key area is built depending on the sequence of keys specified in KEY statements. You must specify keys in the order you want them displayed.

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

Screen attributes for the function key area are set by your system administrator when CA-Easytrieve is installed.

Examples

Code	Meaning
KEY F1	F1 is valid, but nothing is displayed on the screen. You must provide code.
KEY F1 THRU F24	F1 through F24 are valid keys, but nothing is displayed on the screen. You must provide code for all keys.
KEY F1 NAME 'Help'	F1 is valid. F1=Help is displayed on the screen. You must provide code.
KEY F1 F4	F1 and F4 are valid keys, but nothing is displayed on the screen. You must provide code for both keys.
KEY F1 THRU F4, F8	F1, F2, F3, F4, and F8 are valid keys, but nothing is displayed on the screen. You must provide code for all keys.

Special-Named 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 steps illustrate the basic SCREEN activity process that CA-Easytrieve performs and when the special-named procedures, if coded, are executed. The complete process is illustrated later in this chapter.

1. Perform INITIATION procedure.
2. Perform BEFORE-SCREEN procedure.
3. Build the screen using program fields, pending messages, and pending cursor placement.

4. Send the screen to the terminal.
   Terminate and resume the program (if pseudo-conversational).
   Receive the screen from the terminal.
5. Edit input data.
   If any errors, go to step 4.
   If no errors, move the data into the program fields.
6. Perform AFTER-SCREEN procedure.
7. Go to Step 2.
8. When EXIT is requested, perform TERMINATION procedure.

INITIATION

The INITIATION procedure is performed one time during the initiation of the activity. Use INITIATION to perform actions that are only to be executed once, for example, set a field to an initial value or position a file at a specific starting location.

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.

AFTER-SCREEN

The AFTER-SCREEN procedure is performed during each iteration of the activity after the terminal I/O processes. An AFTER-SCREEN procedure can be used to perform complex editing and to update files with data entered on the screen.

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.

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.

⇒ **Reading 1 of Chapter 7 ends here.**

- **If you have completed the tutorial in Chapter 2, continue with the description of Defining Static Working Storage, which is Reading 2 in Chapter 3.**
- **If you branched to this chapter from the tutorial in Chapter 2, go back to Lesson 6 in Chapter 2.**

⇒ **Reading 2 of Chapter 7 starts here.**

Formatting a Screen Item for Display

As discussed in Reading 1, you can use the MASK parameter on the ROW statement to display a numeric field on the screen in a specific format. Two additional parameters on the ROW statement that enable you to customize the display of an item on the screen are JUSTIFY and FILL.

Justifying a Field's Contents

CA-Easytrieve normally displays data exactly as it exists in the field as determined by its definition. Alphanumeric data that does not fill the field size is normally left-justified. Numeric data is normally right-justified. To override default justification, use the JUSTIFY parameter of the ROW statement.

```
ROW [row-number] field-name  +

. . .  +

┌         ⎡RIGHT⎤ ┐
│JUSTIFY  ⎨     ⎬ │ +
└         ⎣LEFT ⎦ ┘

. . .
```

Use of the JUSTIFY parameter is illustrated below:

```
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
```

Filling an Item for Display

Use the FILL parameter of the ROW statement 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.

```
ROW [row-number]  field-name  +

. . .  +

[FILL {'fill-character' | NULL}] +

. . .
```

Varying length fields with FILL NULL 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 to enable 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
```

```
                              Personnel View Utility

Change the employee's name, then press Enter.

         Name  . . . . BRG
```

The user can then insert characters into the field to correct it.

Automatic Editing of Input

CA-Easytrieve automatically edits input data with little or no coding required. In addition to data type validation and mask checking, CA-Easytrieve also performs the following types of edits:

- Uppercasing
- Value checking

You specify these types of editing with additional parameters on the ROW statement:

```
ROW [row-number]  field-name  +

. . .  +

[UPPERCASE] +

[VALUE (literal [THRU literal] [...])] +

. . .
```

The order in which CA-Easytrieve performs editing is:

1. If UPPERCASE is specified for the field, receive the field in all upper-case characters.
2. If a MASK is specified for the field, edit the data against the mask, including data type validation.
3. If a VALUE is specified for the field, edit the data against the value(s).

UPPERCASE

You can specify the UPPERCASE parameter for any field coded on a ROW statement. When UPPERCASE is coded, CA-Easytrieve converts data entered on the screen to upper-case characters as it is received from the terminal.

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 an 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)
```

Cursor Positioning on a Screen

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.
- Execute a CURSOR statement in a screen procedure to specify the field on the screen that receives the cursor on the next display of the screen.

CURSOR Statement

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

Syntax

```
CURSOR AT field-name
```

- *Field-name* refers to a field on a ROW statement within the screen declaration.
- You can use the CURSOR statement only within screen procedures (AFTER-SCREEN, BEFORE-SCREEN, INITIATION, TERMINATION) or procedures performed by screen procedures.

Example

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

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. 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.
2. If a CURSOR statement is not executed, the first field on the screen with the CURSOR attribute receives the cursor.
3. If no field on the screen contains the CURSOR attribute, the first modifiable field on the screen receives the cursor.
4. If there are no modifiable fields on the screen, the first item on the screen receives the cursor.

KEY Statement - Branch Actions and IMMEDIATE Processing

There are three additional parameters you can code on the KEY statement to perform branch actions or to perform IMMEDIATE processing:

- EXIT
- REFRESH
- IMMEDIATE

```
KEY key-name [THRU key-name]...[NAME 'literal'] [EXIT   ] [IMMEDIATE]
                                                [REFRESH]
```

Branch Actions

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 specific steps in the process. The 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

Optionally, you can code EXIT or REFRESH to specify the action taken when a user presses *key-name*. If EXIT or REFRESH is specified, the action is automatically executed by CA-Easytrieve and the AFTER-SCREEN procedure (if any) is not executed.

Specify EXIT to terminate the screen activity after editing and extracting data from screen fields into program fields.

Specify REFRESH to restore the initial screen image by rebuilding it with current values of the program fields. Data in screen fields is edited and extracted into program fields.

If an action is not specified for *key-name*, you can include code for *key-name* in your SCREEN activity procedures.

KEY IMMEDIATE Processing

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.

Specify IMMEDIATE to execute an action, or the AFTER-SCREEN procedure if no action is specified, without editing data in screen fields and moving it into the program fields.

Screen Procedures - 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	9	Terminate the activity

The following steps detail the complete SCREEN activity process that CA-Easytrieve performs.

1. Perform INITIATION procedure, **processing any branch actions.**

2. Perform BEFORE-SCREEN procedure, **processing any branch actions.**
3. Build the screen using program fields, pending messages, and pending cursor placement.
   Clear the internal pending message buffer.
4. Send the screen to the terminal.
   Terminate and resume the program (if pseudo-conversational).
   Receive the screen from the terminal.
5. If KEY-PRESSED is an IMMEDIATE key, go to step 7.
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.
7. **If KEY-PRESSED has an associated branch action, perform it.**
8. Perform AFTER-SCREEN procedure, **processing any branch actions.**
9. Go to Step 2.
10. When EXIT is requested, perform TERMINATION procedure, **processing any branch actions**.

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 (that is, 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.

RESHOW

- RESHOW can be used in an AFTER-SCREEN procedure to redisplay 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 redisplay 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 he requested help.
- 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 redisplay the original screen.

EXIT

- EXIT terminates the SCREEN activity and returns control to the activity from which it was EXECUTEd or, if the current SCREEN activity was not EXECUTEd from another activity, 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.

  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.

⇒ **Reading 2 of Chapter 7 ends here.**

- **Please go on to the beginning of Chapter 8.**

⇒ **Reading 3 of Chapter 7 starts here.**

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 syntax of the IF CURSOR statement is:

Subject **Object**

```
IF  field-name  CURSOR
```

The IF CURSOR statement is included as part of the Field Class Condition. See the *CA-Easytrieve Language Reference Guide* for more information.

The following exhibit illustrates using the IF CURSOR statement to implement a menu:

```
DEFINE VIEW-OPTION   W 1 A
DEFINE EDIT-OPTION   W 1 A
DEFINE DELETE-OPTION W 1 A
DEFINE ADD-OPTION    W 1 A
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 VIEW-OPTION    'View employee'
  ROW   COL 10 EDIT-OPTION    'Edit employee'
  ROW   COL 10 DELETE-OPTION  'Delete employee'
  ROW   COL 10 ADD-OPTION     'Add employee'
  AFTER-SCREEN. PROC
    IF VIEW-OPTION CURSOR
      EXECUTE VIEW-EMPLOYEE
    ELSE-IF EDIT-OPTION CURSOR
      EXECUTE EDIT-EMPLOYEE
    ELSE-IF DELETE-OPTION CURSOR
      EXECUTE DELETE-EMPLOYEE
    ELSE-IF ADD-OPTION 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. The syntax of the IF MODIFIED statement is:

Subject **Object**

```
IF  field-name  MODIFIED
```

The IF MODIFIED statement is included as part of the Field Class Condition. See the *CA-Easytrieve Language Reference Guide* for more information.

Following are the rules for using the IF MODIFIED statement:

- The test for modification determines whether the value of the field actually changed.

  Note: 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 done logically for alphanumeric fields and arithmetically for numeric fields.
- The IF MODIFIED comparison is done after all automatic edit operations have been done for the input data.
- If the screen is received as the result of an IMMEDIATE key, the IF MODIFIED test is always false.

Using the IF MODIFIED test can help you write more efficient programs. For example, you might 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 VERIFY-AND-UPDATE-JOB-CAT
    ELSE
      MESSAGE 'Job category not modified.' LEVEL WARNING
      JOB-CATEGORY = 0
    END-IF
  END-PROC
   ...
```

Overriding System-Defined Attributes and Message Locations

CA-Easytrieve enables you to override display attributes and message locations set by your system administrator at installation by using the DEFAULT statement.

DEFAULT Statement

The DEFAULT statement has two formats. Use Format 1 to specify screen-level overrides of title, field, literal, and function key attributes. Use Format 2 to override message attributes and locations.

Syntax

Format 1

```
DEFAULT {TITLE                                          }
        {FIELD   [ERROR]  ATTR {(attribute-list)}       }
        {LITERAL                                        }
        {KEY                                            }
```

Format 2

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

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

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

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

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

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

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

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

- Specify a list of attribute keywords for the item. See Screen Item Attributes earlier in this chapter for a list of attribute keywords.

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

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

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

Examples

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

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

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

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

Overriding Standard Screen Sizes

As discussed in Reading 1, your system administrator determines the default screen size when CA-Easytrieve is installed at your site. In most instances, you use the default linesize and rowcount for your screens.

However, you can override the default screen size, if necessary, by using the LINESIZE and ROWCOUNT parameters of the SCREEN statement to create "pop-up windows" for your application. The following illustration shows the parameters of the SCREEN statement that define a window:

```
SCREEN [NAME screen-name]  +

  [ROWCOUNT rows]  [LINESIZE columns]  +

  [ROW screen-start-row]  [COL screen-start-col]  +

  [BORDER SINGLE]
```

- ROWCOUNT enables you to override the default number of terminal rows for the screen display.
- LINESIZE permits you to override the default number of terminal columns for the screen display.
- ROW specifies the terminal row where the top of the screen starts.
- COL specifies the terminal column where the left portion of the screen starts.
- BORDER SINGLE specifies that a single line border is to be drawn around the screen.

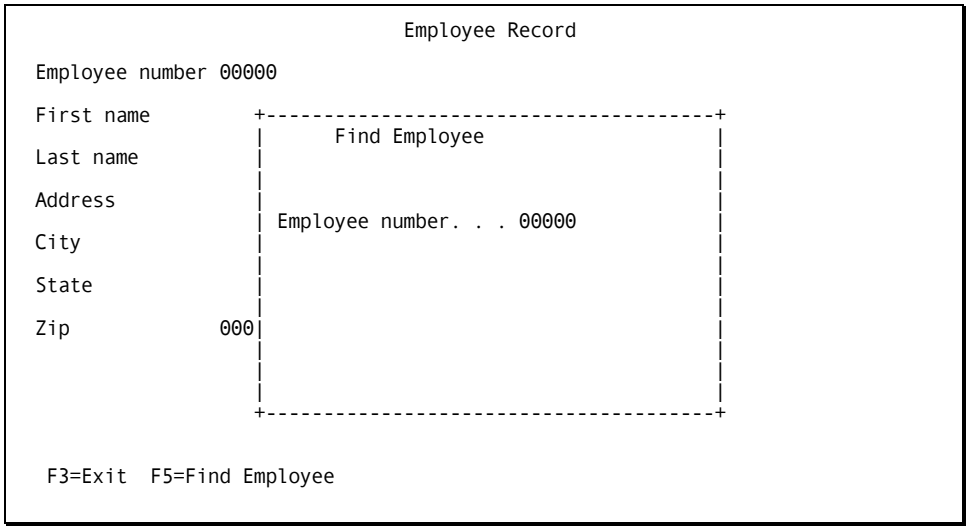
If the LINESIZE and ROWCOUNT for a screen are less than the line size and number of rows on the terminal, the screen is displayed as a pop-up window. Any fields from previous screens that are still displayed are given the ASKIP attribute to prevent data entry on those screens.

Example

The following program code illustrates a full-sized screen executing a pop-up window to prompt the user to enter an employee number to find.

```
SCREEN  NAME SHOW-EMPLOYEE  LINESIZE 80  ROWCOUNT 24
  TITLE 'Employee Record'
  KEY F3 NAME 'Exit' EXIT IMMEDIATE
  KEY F5 NAME 'Find Employee'
  ROW ...
  ...
  AFTER-SCREEN. PROC
    IF KEY-PRESSED = F5
      EXECUTE PROMPT-WINDOW
      READ ...
    ...
  END-PROC
SCREEN  NAME PROMPT-WINDOW  LINESIZE 40  ROWCOUNT 15 +
  ROW 5  COL 20  BORDER SINGLE
  TITLE 'Find Employee'
  KEY ENTER EXIT
  ROW 5 'Employee number. . .' WEMP#
```

When the user presses F5, the screen looks like the screen on the following page.



⇒ **Reading 3 of Chapter 7 ends here.**

- **Please continue with the description of Sequencing a Graph, which is Reading 3 of Chapter 8.**

Chapter

8

Activity Section - Graphs

Introduction

CA-Easytrieve provides a facility for producing bit-mapped presentation graphs with a non-procedural technique very similar to reporting. The graph facility controls the format of the graph by making assumptions based on best-fit. You can easily create: Pie charts, Bar charts, Line graphs, and Scatter diagrams.

Note: Graph processing is available only when using CA-Easytrieve/Workstation.

In this chapter, you'll find:

⇒ **Reading 1**
- Basic structure of a graph program
- Graph display format
- GRAPH statement
- Graph definition statements
- Graph title area, TITLE statement
- Graph work area, VALUE statement
- DRAW statement processing

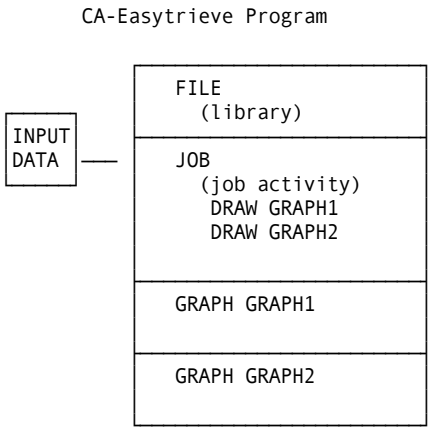
⇒ **Reading 2**
- Graph headings, HEADING statement
- Summing graph values: SUMMARY parameter of the GRAPH statement

⇒ **Reading 3**
- Sequencing a graph: SEQUENCE statement
- Graph display resolution: MODE parameter of the GRAPH statement

⇒ **Reading 1 of Chapter 8 starts here.**

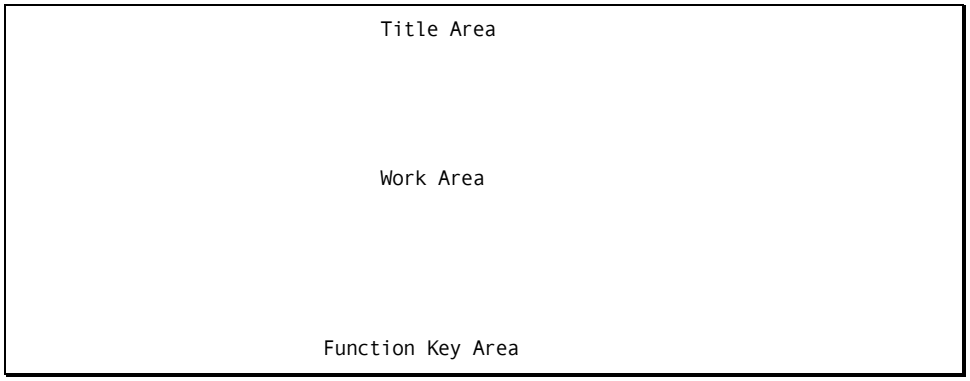
Basic Structure of a Graph Program

With the CA-Easytrieve graph facility, you only need to 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.



Graph Display 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 work area also displays a legend identifying the data.

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 Activity

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.

Syntax

```
                    ⎡STYLE ⎧'PIE'           ⎫⎤
                    ⎢      ⎪'VBAR'          ⎪⎥
                    ⎢      ⎪'SVBAR'         ⎪⎥
                    ⎢      ⎪'HBAR'          ⎪⎥
GRAPH [graph-name]  ⎢      ⎨'SHBAR'         ⎬⎥
                    ⎢      ⎪'LINE'          ⎪⎥
                    ⎢      ⎪'XY'            ⎪⎥
                    ⎢      ⎪'SCATTER'       ⎪⎥
                    ⎣      ⎩style-field-name⎭⎦
```

- Specify the name of the graph. *Graph-name* is optional if there is only one graph coded in a JOB activity.
- STYLE specifies the style of graph to be displayed:
  - Specify PIE to display a pie chart. PIE is the default if the graph style is not coded. The y-value for this graph determines the size of the pie slice. The x-value for this graph determines the category for the y-value.
  - Specify VBAR or SVBAR to display a vertical bar graph. The y-value determines the height of the vertical bar. The x-value determines the category for the y-value.

        - VBAR produces side-by-side vertical bars if multiple y-values are coded on the VALUE statement.
        - SVBAR produces stacked vertical bars if multiple y-values are coded on the VALUE statement.
    - Specify HBAR or SHBAR to display a horizontal bar graph. The y-value determines the length of the horizontal bar. The x-value determines the category for the y-value.
        - HBAR produces side-by-side horizontal bars if multiple y-values are coded on the VALUE statement.
        - SHBAR produces stacked horizontal bars if multiple y-values are coded on the VALUE statement.
    - Specify LINE to display a line graph. The y-value determines the height of the data point on the graph. The x-value determines the category for the y-value.
    - Specify XY to display an XY graph in which values are connected by lines. The y-value determines position of the data point on the y-axis. The x-value determines the position of the data point on the x-axis.
    - Specify SCATTER to display a scatter graph. Values are not connected by lines and a y-axis grid is displayed. The y-value determines position of the data point on the y-axis. The x-value determines the position of the data point on the x-axis.
- *'Style-field-name'* is a field you can define that contains the graph type (one of the style literals described above).

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. These graph definition statements must be coded in the following order:

- SEQUENCE statement — optionally specifies the order of the graph values. You would normally sequence the values by the x-value.
- TITLE statement — defines optional title items and their position on the title line.
- HEADING statement — optionally defines an alternative heading or label for a field.
- VALUE statement — 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.

The SEQUENCE and HEADING statements are discussed in Reading 2.

Graph Title Area

The title area is the first area on a graph. A graph title is optional. You specify the graph title with a TITLE statement coded after the GRAPH statement in the GRAPH subactivity.

TITLE Statement

The TITLE statement specifies the title to be displayed on the graph.

Syntax

```
      ┌┌COL column-number┐ ┌'title-literal' ┐┐
TITLE ├┤                 ├ {                }┤ ...
      └└+offset]         ┘ └title-field-name┘┘
```

- Use COL to display a title at a specific column (*column-number*) on the graph. Graphs are 80 columns wide when text is displayed.
- Title items are separated by one space. Use +*offset* to add additional spaces between title items.
- *'Title-literal'* specifies a character string to be used for the graph title. *Title-field-name* identifies an alphanumeric field to be used for the graph title. The *title-field-name* must be in an active file or W type working storage.

Graph Work Area

The graph work area is built by coding a VALUE statement in a GRAPH activity.

VALUE Statement

VALUE specifies the fields to be used to draw a graph.

Syntax

```
VALUE x-value {y-value [...]}
```

- *X-value* specifies the field or literal to be used for drawing the horizontal axis of the graph. *X-value* must be a numeric field or literal if you are drawing an XY or SCATTER graph.
- *Y-value* specifies the field and or literal to be used for drawing the vertical axis of the graph. Each field or literal must be numeric.
- You can code a maximum of eight *y-values* to produce a multi-series graph. PIE charts are limited to one *y-value*.

- You can use a literal value to count occurrences. For example, the statement VALUE DEPT 1 counts the number of records within each department.
- All graphs are automatically scaled; any data that is too long to be displayed is truncated.
- S type working storage fields cannot be used in a VALUE statement.
- All graphs are de-spooled before REPORTs/SYSPRINT.
- The *y-value* for a PIE graph determines the size of the pie slice. The *x-value* for a PIE graph determines the category for the *y-value*.
- For VBAR (vertical bar) and HBAR (horizontal bar) graphs, the *x-value* determines the category for the y-value.
- For LINE graphs, the *x-value* determines the category for the *y-value*.
- For XY and SCATTER graphs, the *y-value* determines position of the data point on the y-axis. The *x-value* determines the position of the data point on the x-axis.

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.

Syntax

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

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

⇒ **Reading 1 of Chapter 8 ends here.**

- **If you have completed the tutorial in Chapter 2, please go on to the beginning of Chapter 9.**
- **If you branched to this chapter from the tutorial in Chapter 2, go back to the beginning of Chapter 3.**

⇒ **Reading 2 of Chapter 8 starts here.**

Graph Headings

You can inhibit the display of headings on a graph by using the NOHEADING parameter of the GRAPH statement or you can define an alternate heading for a field by coding a HEADING statement in your GRAPH declaration.

Inhibiting Graph Headings

You can use the NOHEADING parameter of the GRAPH statement to suppress the display of X and Y headings on the graph. The default is for X and Y field headings to be displayed on the graph.

GRAPH Statement Syntax

```
GRAPH [graph-name]  +
...  +
[NOHEADING]
```

Defining Alternate Headings

The HEADING statement optionally defines an alternate heading for a field. When defining a field, you specify the default heading. Using the HEADING statement in a report or on a graph enables you to override the default field headings for that graph.

HEADING Statement Syntax

```
HEADING field-name ('heading-literal'...)
```

- *Field-name* specifies a field defined on the VALUE statement of your graph declaration.
- *'Heading-literal'* can be up to 128 characters in length.
- Specify the text for the heading to be displayed for the field on the VALUE statement. Multiple literals, each enclosed within single quotes (' ') and separated by one or more blanks within the parentheses, are displayed on a single line with a space between each literal.
- The HEADING statement is ignored for PIE graphs.

Example

The following code produces a vertical bar graph that displays the sum of the gross pay for each region with user-specified headings:

```
FILE PERSNL F(150)
%PERSNL
JOB INPUT PERSNL NAME DRAW-GRAPH
  DRAW GRAPH1
GRAPH VBAR GRAPH1 SUMMARY MODE('HIGH') TYPE('VBAR')
  SEQUENCE REGION
  HEADING REGION ('Region')
  HEADING PAY-GROSS ('Gross' 'Pay')
  TITLE COL 1 SYSDATE 'GROSS PAY BY REGION' COL 73 SYSTIME
  VALUE REGION PAY-GROSS
```

Summing Graph Values

You can use the SUMMARY parameter of the GRAPH statement to automatically sum all values (y-values) for each category (x-value) before the graph is displayed.

GRAPH Statement Syntax

```
GRAPH [graph-name]  +
. . .  +
[SUMMARY]
```

- For PIE charts, all y-values for each identical x-value are summed producing a slice that is the size of the sum of all of the y-values for this category.
- For VBAR or SVBAR vertical bar graphs, all y-values for each identical x-value are summed, producing a vertical bar that is the sum of all of the y-values for this category.
- For HBAR or SHBAR horizontal bar graphs, all y-values for each identical x-value are summed producing a horizontal bar that is the sum of all of the y-values for this category.
- For LINE graphs, all y-values for each identical x-value are summed producing a data point that is the sum of all of the y-values for this category.
- For XY graphs, all y-values for each identical x-value are summed producing a data point that is the sum of all of the y-values for this x-value.
- For SCATTER graphs, all y-values for each identical x-value are summed producing a data point that is the sum of all of the y-values for this x-value.

⇒ **Reading 2 of Chapter 8 ends here.**

- **Please go on to the beginning of Chapter 3.**

⇒ **Reading 3 of Chapter 8 starts here.**

Sequencing a Graph

You can use the SEQUENCE statement to specify the order of the values on a graph. You can order any graph, based on the content of one or more fields.

SEQUENCE Statement Syntax

```
SEQUENCE field-name [D] ...
```

- *Field-name* identifies a field on which graph values are sequenced. Only one *field-name* is permitted for a graph.
- *Field-name* must be in an active file or W type working storage. Each field must be less than 256 bytes. The fields specified are used as sort keys processed in major to minor order.

  Note: Varying length fields cannot be specified on a SEQUENCE statement.
- An optional D, following a *field-name*, indicates that the field is sequenced into descending order. If you do not code D after a *field-name*, by default the field is sorted in ascending order.
- The fields, used to SEQUENCE a graph, do not have to be part of the displayed graph.

Example

The following illustrates using the SEQUENCE statement in a graph declaration.

```
FILE PERSNL F(150)
%PERSNL
JOB INPUT PERSNL NAME DRAW-GRAPH
  DRAW GRAPH1
GRAPH GRAPH1 SUMMARY
  SEQUENCE BRANCH
  TITLE 'GROSS PAY BY BRANCH'
  VALUE BRANCH PAY-GROSS
```

The above graph declaration produces a pie graph that displays the sum of the gross pay for each branch sequenced by branch.

Graph Display Resolution

You can use the MODE parameter of the GRAPH statement to determine the display resolution of the graph.

GRAPH Statement Syntax

```
                          [     ('LOW'          )]
GRAPH [graph-name] [MODE{ 'HIGH'          }] . . .
                          [     (mode-field-name)]
```

- LOW specifies a resolution of 640 x 200 in black and white. LOW resolution mode is compatible with the CGA, EGA, VGA and MCGA video adapter boards. LOW is recommended if you are going to print the graph.
- HIGH specifies a resolution of 640 x 350 in 16 colors and is compatible with the EGA and VGA video adapter boards.
- *Mode-field-name* is a field you can define that contains LOW or HIGH.
- If not specified, MODE defaults to the highest resolution supported by the video adapter board.

⇒ **Reading 3 of Chapter 8 ends here.**

Chapter

9

System-Defined Fields

Introduction

System-defined fields are fields that CA-Easytrieve automatically defines and maintains internally. You can access these fields in your program to retrieve data that can be useful in processing, or in certain types of error trapping.

This chapter consists of a single reading which describes the four categories of system-defined fields that CA-Easytrieve provides:

⇒ **Reading 1**

- General purpose fields
- File processing fields
- Report processing fields
- Screen processing fields

⇒ **Reading 1 of Chapter 8 starts here.**

General Purpose Fields

CA-Easytrieve automatically provides the system-defined fields listed below for general use.

SYSDATE

SYSDATE is a read-only field that contains the system date at the start of CA-Easytrieve execution. The DATE option of the Options Table (see your *CA-Easytrieve Language Reference Guide*) determines the format of the date. Normally, a slash (/) separates the month, day, and year components of the date (for example, MM/DD/YY).

SYSDATE-LONG

SYSDATE-LONG is a read-only field that contains the system date at the start of CA-Easytrieve execution and is similar to SYSDATE, except that it includes the century (for example, MM/DD/YYYY).

SYSTIME

SYSTIME is a read-only field that contains the system time at the start of CA-Easytrieve execution. Normally, a colon (:) separates the data into hours, minutes, and seconds (for example, HH:MM:SS).

RETURN-CODE

RETURN-CODE is a field whose contents are returned to the operating system in register 15 when CA-Easytrieve terminates. RETURN-CODE is initialized to zero, but you can set it to any value. RETURN-CODE applies only to MVS systems.

UIBFCTR

When processing an IMS/DLI database in a CICS environment, UIBFCTR contains the values from the UIBFCTR fields in the CICS UIB. See the *CICS Application Programmer's Reference Manual* for a description of the UIBFCTR fields.

UIBDLTR

When processing an IMS/DLI database in a CICS environment, UIBDLTR contains the values from the UIBDLTR fields in the CICS UIB. See the *CICS Application Programmer's Reference Manual* for a description of the UIBDLTR fields.

UIB-ADDRESS

When processing an IMS/DLI database in a CICS environment, UIB-ADDRESS contains the address of the CICS UIB. It only contains the UIB- ADDRESS following the execution of a Format 5 DL/I statement. See the *CICS Application Programmer's Reference Manual* for a description of the UIB.

File Processing Fields

CA-Easytrieve automatically provides the system-defined fields listed below for each of your files. These fields are stored as part of working storage but can be qualified by file-name. As working storage fields, they are not subject to invalid file reference errors.

RECORD-LENGTH

RECORD-LENGTH is a field with zero decimal places 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 actual data. CA-Easytrieve automatically adjusts the field to account for the four-byte record-control word and four-byte block-control-word. For variable-length files, assign the length of the record to the RECORD-LENGTH field before the PUT or WRITE statement is executed.

For SQL files, RECORD-LENGTH contains the sum of the maximum lengths of all fields in the file. For CA-IDMS and IMS/DLI files, RECORD-LENGTH contains the sum of the maximum lengths of all records in the file.

RECORD-COUNT

RECORD-COUNT is a read-only field with zero decimal places that contains the number of logical I/O operations performed on a file.

For CA-IDMS and IMS/DLI files, only automatic input increments RECORD-COUNT.

FILE-STATUS

FILE-STATUS is a read-only field that contains the results of the most recent I/O operation on a file. FILE-STATUS is available when you code STATUS on the I/O statement. If you do not code STATUS, an appropriate error message is generated. The error message contains one of these codes.

For CA-IDMS files using automatic input, FILE-STATUS contains IDMSSTATUS. For IMS/DLI files, FILE-STATUS contains the status code from the PCB.

FILE-STATUS codes and their meanings are:

0000 Operation successful.

Explanation: This is not an error condition. It indicates that the last I/O operation was successful. No additional information is required.

0004 End of file.

Explanation: This is not an error condition. It indicates that the file position has been moved beyond the last record in the file.

Cause: This condition occurs following a GET statement when the current record is the last record in the file. It can occur for SEQUENTIAL, INDEXED, and RELATIVE files.

Following a GET PRIOR statement, this condition could also indicate the beginning of a file.

0008 Record with a duplicate alternate key exists.

Explanation: This is not an error condition. It indicates that the key of this record matches the key of the record that follows it in the sequential order of this file.

Cause: This condition can occur following a GET or READ statement for an INDEXED file that does not have unique keys.

Following a GET statement, this condition indicates that at least one more record with a matching key is waiting to be processed.

Following a READ statement, this condition indicates that there is at least one more record in the file with a matching key (a GET statement must be used to retrieve any remaining records).

In CICS/VS, MVS (batch and TSO), and CMS/OS, an INDEXED file can have non-unique keys if the associated data set is a VSAM PATH and the auxiliary index data set was defined with non-unique keys.

Note: There is no alternate or primary index on the workstation, in BETRIEVE, or in ISAM/VSAM, therefore, a status code of 8 is never encountered. However, if you move the application to the mainframe, you should test for this condition.

0012 Duplicate key.

Explanation: This error condition indicates that an attempt was made to store a record with a duplicate key, or there is a duplicate record for an alternate index with the Unique Key option.

Cause: This condition can occur following a PUT or WRITE ADD statement for an INDEXED file, or a PUT statement for a RELATIVE file.

For an INDEXED file, it indicates that the key of the record matches the key of a record already present in the file. For a RELATIVE file, it indicates that the slot designated by the relative record number already contains a record (the slot is not empty).

This condition can also occur following a WRITE UPDATE statement for a SEQUENTIAL or INDEXED file. It indicates that:

- There is at least one alternate index associated with this file.
- The alternate index was defined with the unique key and the upgrade option.
- The updated record caused a duplicate key condition to occur when the alternate index was updated.

0016 Record not found.

Explanation: This error condition indicates that the record designated by the KEY parameter is not found in the file.

Cause: This condition can occur following a READ or POINT statement for an INDEXED or RELATIVE file. For an INDEXED file, it indicates that no record in the file matches the key specified by the statement. For a RELATIVE file, this condition indicates that the slot designated by the relative record number is empty.

0020 Record locked.

Explanation: This error condition indicates that an attempt was made to access or update a record that has a lock placed on it by another process.

Cause: This condition is only possible on the workstation.

0024 Logical or physical error condition.

Explanation: This error condition indicates that a logical or physical error condition was detected by the access method routines used to access the file. The specific cause of the error is displayed in a runtime abend message. See Appendix A of the *CA-Easytrieve/Online User Guide* for a list of the feedback codes.

PATH-ID

For CA-IDMS and IMS/DLI files, PATH-ID is field that contains the ID value of the lowest record retrieve in a path, using the RETRIEVE statement. See the *CA-Easytrieve Programmer Guide* for more information.

IDMSCOM

IDMSCOM contains a set of fields defined for the CA-IDMS Communications Block. See the *CA-Easytrieve Programmer Guide* for more information.

SLC

SLC contains a set of fields defined for a logical record communications block. See the *CA-Easytrieve Programmer Guide* for more information.

SQLCA

SQLCA contains a set of fields defined for the SQL communications Block. See the *CA-Easytrieve Programmer Guide* for more information.

Report Processing Fields

CA-Easytrieve automatically provides the system-defined fields listed below for report generating. These fields are stored as part of working storage and are read-only.

LINE-COUNT

LINE-COUNT contains the number of lines printed on the page.

LINE-NUMBER

LINE-NUMBER contains the number of the line being printed within the line group.

PAGE-COUNT

PAGE-COUNT contains the number of pages printed.

PAGE-NUMBER

PAGE-NUMBER contains the number of the page being printed.

TALLY

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 level.

TALLY is a field with zero decimal places. This definition is used for calculations contained within report procedures. The TALLYSIZE parameter of the REPORT statement 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 determining which control break is currently active. The value in LEVEL indicates the control break level and varies from 0 to n based on the number of field names on the CONTROL statement of the associated report. LEVEL contains the logical position number of the controlling field name. This value also applies to FINAL whether it is coded or not.

LEVEL Example

```
LEVEL = 1 on TERRITORY breaks.
LEVEL = 2 on REGION breaks.
LEVEL = 3 on AREA breaks.
LEVEL = 4 final break.

CONTROL  FINAL  NOPRINT  AREA  REGION  TERRITORY
           4               3      2       1

CONTROL  AREA  REGION  TERRITORY
     4     3      2         1
```

BREAK-LEVEL

BREAK-LEVEL indicates the highest field in the break.

Screen Processing Fields

CA-Easytrieve automatically provides the system-defined fields listed below for screen applications. These fields are stored as part of working storage and are read-only.

KEY-PRESSED

KEY-PRESSED is a field that contains a value representing the most recent terminal key pressed by the terminal user.

CA-Easytrieve automatically defines symbolic names that correspond to values for the most common keys. Only keys with symbolic names can be used on a KEY statement.

Terminal Key	Symbolic Name	Constant Value
Unknown		0
Enter	ENTER	1
Clear	CLEAR	11
PA1 thru PA3	PA1 thru PA3	12 thru 14
PF1 thru PF24	F1 thru F24	21 thru 44
F1 thru F12	F1 thru F12	21 thru 32
Test Request		220
Op ID card Reader		222
Magnetic Slot Reader		223
Trigger Action		224
Structured Field		230
Clear Partition		231
Read Partition		232
No Aid Generated		255

TERM-COLUMNS

TERM-COLUMNS is a field that contains the maximum number of columns the screen supports. You can test TERM-COLUMNS to execute a SCREEN activity designed specifically for the terminal being used.

TERM-ROWS

TERM-ROWS is a field that contains the maximum number of rows the screen supports. You can test TERM-ROWS to execute a SCREEN activity designed specifically for the terminal being used.

TERM-NAME

TERM-NAME is a field that contains the terminal identification. This field is set only in CICS environments.

SYSUSERID

SYSUSERID is a field that identifies the terminal user. In CICS, this field is retrieved from the EIB.

⇒ **Reading 1 of Chapter 9 ends here.**

- **Please go back to the beginning of Chapter 3.**

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z