

# **CA-Easytrieve<sup>®</sup>/**

# **Plus**

# **For OS/390, VM, VSE**

## **User Guide**

## **6.2**

**COMPUTER<sup>®</sup>**  
**ASSOCIATES**  
*Software superior by design.*

SP3

Release 6.2, May 1996  
Updated: April 1999

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

THIS DOCUMENTATION MAY NOT BE COPIED, TRANSFERRED, REPRODUCED, DISCLOSED OR DUPLICATED, IN WHOLE OR IN PART, WITHOUT THE PRIOR WRITTEN CONSENT OF CA. THIS DOCUMENTATION IS PROPRIETARY INFORMATION OF CA AND PROTECTED BY THE COPYRIGHT LAWS OF THE UNITED STATES AND INTERNATIONAL TREATIES.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

THE USE OF ANY PRODUCT REFERENCED IN THIS DOCUMENTATION AND THIS DOCUMENTATION IS GOVERNED BY THE END USER'S APPLICABLE LICENSE AGREEMENT.

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

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227.7013(c)(1)(ii) or applicable successor provisions.

© 1996-2000 Computer Associates International, Inc., One Computer Associates Plaza, Islandia, New York 11749. All rights reserved.

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



# Contents

---

## Chapter 1: About This Guide

Purpose and Audience .....	1-1
Organization .....	1-1
Other CA-Easytrieve/Plus Publications .....	1-2
Documentation Conventions .....	1-3
Variable Parameters .....	1-4

## Chapter 2: Overview

Before You Begin .....	2-1
Things You Should Know .....	2-1
Reading This Guide .....	2-1
Program Examples .....	2-3
Flowchart Symbols .....	2-3
Introduction to CA-Easytrieve/Plus .....	2-4
Benefits .....	2-4
Capabilities .....	2-4
Current Technology .....	2-8
Environments .....	2-8
Structure of a CA-Easytrieve/Plus Program .....	2-8
Environment Definition Section .....	2-9
Library Definition Section .....	2-9
Activity Definition Section .....	2-9
Sample Program .....	2-11

---

## Chapter 3: Standard Reporting with CA-Easytrieve/Plus

Reports Made Easy-A Tutorial .....	3-1
Reading This Tutorial .....	3-1
Lesson 1: Your First CA-Easytrieve/Plus Report Program .....	3-2
Report Your Program Creates .....	3-3
One Statement at a Time .....	3-3
FILE Statement .....	3-3
DEFINE Statement .....	3-4
Reviewing the Library Section .....	3-6
Lesson 2: Expanding Your First Report Program .....	3-7
JOB Statement .....	3-7
A Look At Logic .....	3-8
CA-Easytrieve/Plus Working Storage .....	3-10
CA-Easytrieve/Plus LINE Statement .....	3-11
Review of Job Activities .....	3-13
Lesson 3: Printing CA-Easytrieve/Plus Reports .....	3-13
Editing Your Report Output .....	3-14
Field Headings .....	3-16
Reviewing PRINT, MASK, and HEADING .....	3-17
Lesson 4: Report Declarations .....	3-18
REPORT Statement .....	3-19
Report Definition Statements .....	3-19
SEQUENCE Statement .....	3-20
CONTROL Statement .....	3-21
SUM Statement .....	3-22
TITLE Statement .....	3-22
HEADING Statement .....	3-23
LINE Statement .....	3-24
Reviewing Report Declarations .....	3-24
Lesson 5: Making Your Job Easier with Macros .....	3-25
Lesson 6: Diagnosing An Error .....	3-27
Summing Things Up .....	3-28

## Chapter 4: Library Section-Describing and Defining Data

Introduction .....	4-1
Syntax Rules .....	4-2
Statement Area .....	4-2
Multiple Statements .....	4-2
Comments .....	4-2

---

Continuations	4-3
Words and Delimiters	4-3
Keywords	4-4
Multiple Parameters	4-4
Field Names	4-4
Labels	4-5
Alphabetic Literals	4-5
Numeric Literals	4-5
Hexadecimal Literals	4-5
Identifiers	4-6
Arithmetic Operators	4-6
Describing Files and Fields	4-6
Defining Data	4-6
FILE Statement	4-7
DEFINE Statement	4-8
FILE Statement Revisited	4-19
Virtual File Manager (VFM)	4-19
EXIT Parameter	4-20
COPY Statement	4-21

## Chapter 5: Activity Section-Processing and Logic

Introduction	5-1
JOB Activities	5-2
JOB Statement	5-2
Conditional Expressions	5-3
Calculations	5-7
Assignment Statement	5-8
MOVE Statement	5-10
MOVE LIKE	5-11
DO WHILE/END-DO Statements	5-11
GOTO Statement	5-12
STOP Statement	5-13
SORT Statement	5-14
User Procedures (PROCs)	5-16
START/FINISH Procedures	5-18
Processing Tables	5-18

---

## Chapter 6: Activity Section-Input and Output

Introduction .....	6-1
Automatic Input and Output .....	6-2
Automatic Input With the JOB Statement.....	6-2
Printing Reports .....	6-2
User Controlled Input and Output .....	6-5
Sequential File Processing .....	6-5
Random Access Processing .....	6-10

## Chapter 7: Activity Section-Reporting

Introduction .....	7-1
Standard Reports .....	7-2
Titles .....	7-3
Headings .....	7-4
Line Group .....	7-5
Line Item Positioning .....	7-5
Report Processing .....	7-6
REPORT Statement .....	7-6
REPORT Statement Example .....	7-8
Report Definition Statements .....	7-9
Label Reports .....	7-15
Label Format .....	7-16
Testing Aid Parameters .....	7-17
Format Determination Parameters.....	7-19
DTLCTL .....	7-20
SUMCTL .....	7-21
SUMMARY Reports.....	7-25
DTLCOPY .....	7-25
Summary Files .....	7-26
Multiple Reports .....	7-28
Multiple Reports to Single Printer .....	7-28
Multiple Reports to More Than One Printer .....	7-29
Report Procedures (PROCs) .....	7-30

---

## Chapter 8: Macros

Introduction .....	8-1
Using Macros .....	8-2
What is a CA-Easytrieve/Plus Macro? .....	8-2
Invoking Macros .....	8-2
PERSNL File Field Definitions .....	8-3
Macro Invocation Statement .....	8-4
Macro Nesting .....	8-4
Creating and Storing Macros .....	8-5
Defining Macros .....	8-5
Prototype Statement .....	8-5
Macro Body .....	8-6
Macro Termination Command .....	8-6
Storing Macros .....	8-6
Parameter Substitution in Macros .....	8-7
Positional Parameters .....	8-7
Keyword Parameters .....	8-8
Prototype Statement .....	8-8
Prototype Statement Parameter Examples .....	8-8
Macro Invocation Statement .....	8-9
Parameter Substitution Examples .....	8-10
Positional Parameters .....	8-10
Keyword Parameters .....	8-11
Rules for Substituting Parameters .....	8-12
Ampersands (&) and Periods (.) in Macros .....	8-13
Instream Macros .....	8-13
Operation .....	8-14

## Chapter 9: Programming Techniques

Introduction .....	9-1
Abnormal Termination .....	9-1
Diagnostic Messages .....	9-2
Syntax Errors .....	9-2
Execution Errors .....	9-2
Statement Listing .....	9-2
PARM Statement and System Options Table .....	9-3
PARM Statement Parameters .....	9-4
PARM Examples .....	9-6

---

System-Defined Fields .....	9-6
General Purpose Fields .....	9-7
File Processing Fields .....	9-7
Report Processing Fields.....	9-8

## Index

# About This Guide

---

## Purpose and Audience

The purpose of this guide is to teach new users of CA-Easytrieve/Plus how to write CA-Easytrieve/Plus programs. Our goal is to put the simplicity and power of this language to work for you so that you can become productive immediately.

By the time you finish Chapter 3, you will be writing standard reports with CA-Easytrieve/Plus. For many of you, this might be all you ever want to do. For the rest of you, there is much, much more.

## Organization

This *CA-Easytrieve/Plus User Guide* is divided into several chapters:

- “Overview” gives you a complete overview of CA-Easytrieve/Plus and this *User Guide*. It tells you what CA-Easytrieve/Plus can do, where it can do it, and what the program looks like.
- “Standard Reporting with CA-Easytrieve/Plus” takes you, tutorial style, through the process of creating a CA-Easytrieve/Plus report.
- “Library Section – Describing and Defining Data” tells you how to describe the files for processing. It includes special features that CA-Easytrieve/Plus provides to make this job easier.
- “Activity Section – Processing and Logic” describes the basics of writing an application program with CA-Easytrieve/Plus. Everything from IF statements to table processing.
- “Activity Section – Input and Output” teaches you how to handle (or let CA-Easytrieve/Plus handle) input and output files, from automatic I/O to random access of VSAM files.
- “Activity Section – Reporting” explains all of the things you did not learn about reporting in Chapter 3.
- “Macros” teaches you how to create and invoke CA-Easytrieve/Plus macros.

- “Programming Techniques” describes diagnostic messages and CA-Easytrieve/Plus debugging facilities.
- The Index provides a listing to facilitate references to terms and procedures.

## Other CA-Easytrieve/Plus Publications

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

<b>Name</b>	<b>Contents</b>
<i>CA-Easytrieve/Plus Installation Guide</i>	Describes the process of installing the CA-Easytrieve/Plus system in all environments. You can find the most current step-by-step procedures for installing CA-Easytrieve/Plus in your environment on the product tape.
<i>CA-Easytrieve/Plus Reference Guide</i>	Contains descriptions of all product features and functions and summaries of each CA-Easytrieve/Plus version.
<i>CA-Easytrieve/Plus CA-Activator Supplement</i>	Explains how to install and maintain CA-Easytrieve/Plus using CA-ACTIVATOR.
<i>CA-Easytrieve/Plus Application Guide</i>	Describes basic syntax (a subset of the syntax in the <i>CA-Easytrieve/Plus Reference Guide</i> ) and operation, and provides a series of actual applications, from single examples to full systems. The <i>Application Guide</i> is an excellent tool for the business-oriented professional.
<i>CA-Easytrieve/Plus Extended Reporting Facility Guide</i>	Describes support of extended reporting capabilities for Impact Dot, Ink Jet, and Electro Photographic printers.
<i>CA-Easytrieve/Plus Interface Option Guides</i>	Short guides available for users of various system options. They consist of guides for IMS/DLI processing, CA-IDMS and IDD processing, TOTAL processing, SQL processing, CA-Datcom/DB processing, SUPRA processing, and other CA-Easytrieve/Plus options.

## Documentation Conventions

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

<b>Notation</b>	<b>Meaning</b>
{braces}	Mandatory choice of one of these entries.
[brackets]	Optional entry or choice of one of these entries.
(OR bar)	Choice of one of these entries.
(parentheses)	Multiple parameters must be enclosed in parentheses.
...	Ellipses indicate that you can code the immediately preceding parameters multiple times.
CAPS	All capital letters indicate a keyword, name, or field used in a program example.
lowercase	Lowercase letters represent variable information in statement syntax. If the same variable types recur in a statement, they are made unique by adding a numeric suffix, such as literal-2.

## Variable Parameters

<b>Parameter</b>	<b>Meaning</b>
field-name	Data field defined in your program.
file-name	Unique file name defined in the library section of your program.
index name	Name of an INDEX data item.
integer	Numeric literal (a whole number greater than zero).
job-name	Name of a JOB activity.
letter	Single alphabetic character (such as an edit mask identifier).
literal	Text string enclosed in quotes or a numeric constant.
proc-name	Name of a procedure.
program-name	Name of a program written in a language other than CA-Easytrieve/Plus (such as COBOL or Assembler).
record-name	Name of an IMS/DLI or a CA-IDMS entity.
report-name	Name of a REPORT.
sort-name	Name of a SORT activity.

## Before You Begin

### Things You Should Know

As stated in Chapter 1, “About This Guide,” the purpose of this documentation is to teach you how to write CA-Easytrieve/Plus programs. We are, however, assuming 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.

Because CA-Easytrieve/Plus 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 sites. It is impossible to address the specifics of all the operating environments CA-Easytrieve/Plus can run under.

We assume 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 enter data and execute programs at your site, then this guide can teach you what you must know about CA-Easytrieve/Plus.

### Reading This Guide

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

#### Reading One

The section you are reading now and the tutorial in Chapter 3 should be read in their entirety by everyone. The tutorial enables readers who have an interest in more detail to branch off to first level readings of appropriate sections at various intervals.

Readers are always directed to return to the tutorial after they complete the material at the end of such a branch. The goal for the first pass through the guide is to read Chapters 2 and 3 and the first level readings of all the rest of the chapters. Doing so gives you a good understanding of CA-Easytrieve/Plus basics and lets you perform the following tasks:

- Write a complete CA-Easytrieve/Plus program using automatic input and output features.
- Generate standard reports.
- Perform calculations and use conditional expressions.
- Perform simple macro invocation.
- Read and understand diagnostic messages.

## Reading Two

The second reading begins at level 2 of Chapter 4 and continues with level 2 of Chapters 5 through 9. After completing this second reading of the *CA-Easytrieve/Plus User Guide*, you should be able to perform these tasks:

- Write slightly more complex CA-Easytrieve/Plus programs using programmer-controlled input and output commands.
- Generate label reports.
- Perform data assignments and moves and use loops and branching in program logic.
- Create macros.
- Perform basic debugging techniques.

## Reading Three

The third and final reading, like reading two, continues your journey through Chapters 4 through 9. It teaches you some of the more sophisticated commands and techniques available with CA-Easytrieve/Plus. 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.
- Use positional and keyword parameters of the MACRO prototype statement.

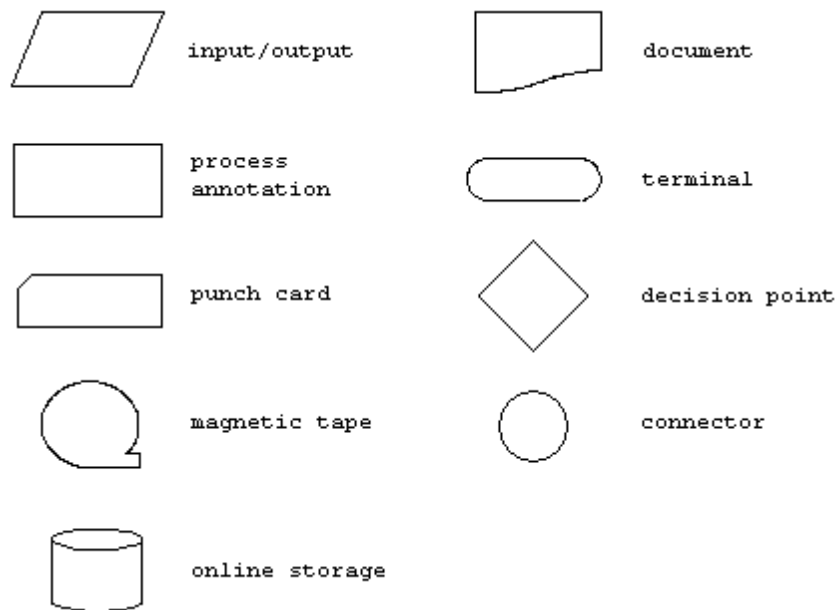
## Program Examples

Most of the program examples in this guide use an input file named PERSNL. This file is made available when CA-Easytrieve/Plus is installed so that new users can enter data and execute the program examples shown in the documentation. (See the *CA-Easytrieve/Plus Installation Guide* for information on sample files provided with CA-Easytrieve/Plus.)

Many of the program output examples, such as reports, were edited or shortened for illustrative purposes. Reports you produce yourself from the PERSNL file can be much longer than the ones shown in this guide.

## Flowchart Symbols

The following flowchart symbols illustrate the same concepts throughout this document.



## Introduction to CA-Easytrieve/Plus

### Benefits

CA-Easytrieve/Plus 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 with ease. Its comprehensive logic and processing facilities give the experienced data processor the capabilities to perform complex programming tasks.

Unlike other programming languages that provide ease of use while sacrificing flexibility, CA-Easytrieve/Plus provides the best of both worlds.

### Capabilities

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

### File Access

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

- Accepts up to 890 input or output files.
- Synchronizes file processing (based on keys) of an unlimited number of files, including matched conditions and duplicate checking. This reduces complex matching logic down to one statement.
- Tests for file availability and current record count.
- Prints statistics on files used, including number of records processed and attributes of each file.
- Provides in-core binary search of external or instream table files.
- Prints 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.

## Double-Byte Character Set (DBCS) Support

CA-Easytrieve/Plus supports Double-Byte Character Set (DBCS) character representations. See the *CA-Easytrieve/Plus Reference Guide* for more information on DBCS support.

## Field Definition

The CA-Easytrieve/Plus methods of defining 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.
- Supporting alphanumeric field types, containing both EBCDIC and DBCS format data, plus a MIXED field type for those fields that contain a mixture of both EBCDIC and DBCS format characters.
- Providing flexible edit masks for report formats or displaying data, including blank-when-zero, automatic DBCS conversion, and hex display.
- Establishing EBCDIC, DBCS, and MIXED initial values for working storage fields.
- Providing default report headings to enhance standards.
- Permitting multiple use of field definitions with the COPY keyword, reducing coding and maintenance.

## Logic Process

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

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

## File Output

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

- Loading and updating files, including VSAM, IMS/DLI, IDMS, and SQL.
- Saving report extract work files for subsequent use.
- Providing a selective hex dump of a file or specific fields.

## Report Output

The CA-Easytrieve/Plus reporting features 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/Plus:

- Produces unlimited reports from a single pass of the data.
- Automatically formats reports including where character output sizes vary due to different data types (EBCDIC and DBCS formats) and font specifications (see the *CA-Easytrieve/Plus Extended Reporting Facility Guide*).
- 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 your REPORT statements require.
- Generates reports to separate logical printers or other output media.
- Provides control break level access for special logic processing, which is useful when only certain report lines are generated for certain specific levels of control breaks.
- Provides specialized report procedures for user flexibility, such as: BEFORE/AFTER-LINE, ENDPAGE, TERMINATION, BEFORE/AFTER-BREAK, REPORT-INPUT.
- Permits explicit positioning of print layout for pre-printed forms.

## Virtual File Manager (VFM)

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/Plus keywords. VFM's own data management techniques ensure its operating efficiency standards, including:

- Maintaining 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 use.
- Releasing and recovering occupied space as the virtual file is read back into your program.
- Automatically spooling files containing report information created as a result of sequenced reports or multiple reports in the same activity.

## Debugging Capabilities

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

- Providing an error analysis report that pinpoints most errors immediately, including the source statement number in error and a FLOW table of what statements executed in what series.
- Providing optional data processing oriented displays, such as data maps (DMAPs) and program maps (PMAPs).
- Trapping invalid file references during execution to prevent a system dump.

## Current Technology

CA-Easytrieve/Plus represents the maximum in efficiency because it was developed with the latest in programming technology, including:

- Mapping programs in 4K segments.
- Mapping working storage on double-word boundary.
- Providing a one-pass compiler.
- Directly generating the object code.
- Providing PUSH/POP facilities for MACRO.
- Providing security on VSAM, IMS, and IDMS use.
- Automatic EBCDIC to DBCS conversion facilities and user exits for implementing Phonetic Translation routines.

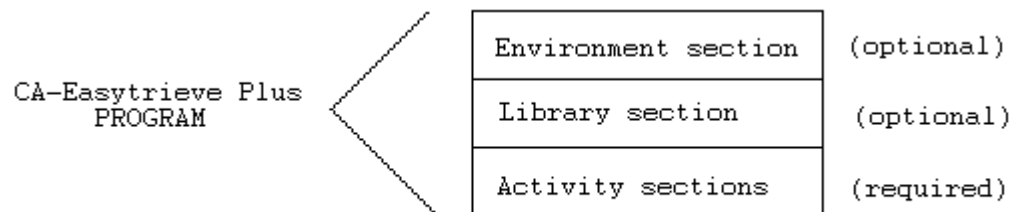
## Environments

CA-Easytrieve/Plus operates on the IBM 370, 30xx, 43xx, and compatible processors in the DOS/VSE, OS/VS, and VM/CMS environments. Under TSO, CMS, and ICCF, CA-Easytrieve/Plus can run interactively for data inquiry, analysis, and reporting. The output can be returned back to your terminal screen or routed to a printer.

## Structure of a CA-Easytrieve/Plus Program

Before beginning the tutorial in Chapter 3, it is helpful to have a basic understanding of how a CA-Easytrieve/Plus program is structured.

Each CA-Easytrieve/Plus program can contain an *environment definition* 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 Definition Section

The environment definition section establishes parameters for the program. This section lets you override standard CA-Easytrieve/Plus options and to choose a mode of operation. The environment definition section is not required for most of the examples in this guide. For a complete description of the environment definition section, see the *CA-Easytrieve/Plus Reference Guide*.

## Library Definition Section

The library definition section describes the data the program is to process. It describes data files and their associated fields and 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 Definition Section

The activity definition section is the only mandatory section of your program. There are two types of activities – JOB and SORT. You can code any number of JOB or SORT activities in any order.

- JOB activities read information from files, examine and manipulate data, write information to files, and initiate printed reports.
- SORT activities create sequenced files. (Files with records in alphabetical or numerical order.)

You can code one or more procedures (PROCs) at the end of each activity. Procedures are separate modules of program code that you use to perform specific tasks and are described in Chapter 5, “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 in a REPORT subactivity (REPORT PROCs) immediately after the REPORT subactivity where you use them.

The following example shows some of the CA-Easytrieve/Plus keywords and other items in the chapters where they are usually located. It gives the general order of CA-Easytrieve/Plus statements in a program.

Environment section	PARM ...
Library section	FILE ... DEFINE ... ...
Activity section	JOB (statements) (job procedures) REPORT (report procedures) SORT (sort procedures) ...

## Sample Program

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

```

FILE PERSNL FB(150 1800)          }
  NAME  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 NAME EMP# GROSS     }

```

The above program produces the following output (as is the case in many of the output examples in this guide, it was edited for illustrative purposes).

```

11/02/88          PERSONNEL REPORT EXAMPLE-1          PAGE    1

      DEPT      NAME      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 already realize that CA-Easytrieve/Plus 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/Plus reports.

We hope you enjoy programming with CA-Easytrieve/Plus; in fact, we are sure you will. Good Luck!



# Standard Reporting with CA-Easytrieve/Plus

---

## Reports Made Easy-A Tutorial

You need information and you need it now. The data processing department is backlogged two weeks. You know where your data is and you were just given access to a programming language called CA-Easytrieve/Plus. What do you do? That is what this tutorial is all about.

In this tutorial, you will learn:

- How to quickly create a report using CA-Easytrieve/Plus.
- How to redefine your report to suit your needs.
- How CA-Easytrieve/Plus macros can save you programming time.
- What to do when you encounter an error.

## Reading This Tutorial

This tutorial is divided into six lessons. At the end of each lesson, you are given an opportunity to go 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 finish you can write a CA-Easytrieve/Plus report program.

## Tutorial Lessons

The six lessons in this section correspond (generally) to Chapters 4 through 9 of this guide. The topics covered, in order, are:

### Lesson 1

CA-Easytrieve/Plus library section, including FILE and DEFINE statements.

Lesson 2

CA-Easytrieve/Plus activity section, including JOB and IF statements. Also includes a look at working storage fields defined in the library section.

Lesson 3

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

Lesson 4

CA-Easytrieve/Plus activity section, including the REPORT statement and report definition statements.

Lesson 5

Using a CA-Easytrieve/Plus macro to replace the library section.

Lesson 6

Responding to an error in your CA-Easytrieve/Plus program.

## Lesson 1: Your First CA-Easytrieve/Plus Report Program

To start out, we show you a CA-Easytrieve/Plus 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)
  NAME      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 NAME EMP# GROSS
```

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

## Report Your Program Creates

The program we just showed you produces a report similar to the one shown below:

11/02/88		PERSONNEL REPORT EXAMPLE-1		PAGE	1
DEPT	NAME	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 a simple edited display of fields from a file of employees named PERSNL. (As we mentioned in the "Overview" chapter, this sample file is provided with CA-Easytrieve/Plus. Ask your system administrator where it is stored at your site.) It is a good starting point for describing some of the most important CA-Easytrieve/Plus keywords.

## One Statement at a Time

Let us go on by examining the cause and effect relationship between our program and its report, one statement at a time. We start out with the CA-Easytrieve/Plus library section (described in the "Overview" chapter).

We are going to keep it simple; if you start getting bored and feel you want more, remember you are directed to more detailed information at the end of this lesson.

## FILE Statement

The first line of our program looks like this:

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

This line contains the CA-Easytrieve/Plus FILE statement. You must include a FILE statement for every file you use as input to your program. It tells CA-Easytrieve/Plus 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. In our example, that name is PERSNL.

The rest of line 1 is optional. It tells CA-Easytrieve/Plus 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 in parentheses in CA-Easytrieve/Plus. Since record length (150) and blocksize (1800) are mandatory subparameters of FB, we include them in parentheses.

## DEFINE Statement

There are four DEFINE statements in our program:

```
NAME      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 do not see the word DEFINE in the above lines, but it is implied. We could have written these lines as:

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

You can use the DEFINE statement 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 just implied. We cover this more thoroughly in the “Library Section-Describing and Defining Data” chapter.

### Using DEFINE to Describe Fields

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

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

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



## Data Type

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

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

## Number of Decimal Positions

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

```

                Decimal
                Positions
                -----
GROSS  94  4  P    2
    
```

When this field prints 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 covered so far, FILE and DEFINE, comprise the bulk of the CA-Easytrieve/Plus library section. These two statements define the library of data CA-Easytrieve/Plus uses as input to any processing activities.

- FILE tells CA-Easytrieve/Plus about the data file you are accessing (or creating).
- DEFINE tells CA-Easytrieve/Plus which fields to use from the file.

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

## For More Information

If you want to add more detail to your understanding of the library section, turn to Chapter 4, “Library Section-Describing and Defining Data.”

If you are content with what you learned so far, continue with Lesson 2.

## Lesson 2: Expanding Your First Report Program

In Lesson 1 of this tutorial, we showed you a complete CA-Easytrieve/Plus program and described part of it, called the library definition section. We still have not explained how your data is processed to produce a report.

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

First, let us talk about the JOB statement.

### 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/Plus 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.

### Input to a JOB Activity

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

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

Although CA-Easytrieve/Plus does give 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) should be *automatically* made available to your program. It is like saying, “I want to use this file” and then letting CA-Easytrieve/Plus do the rest.

In fact, CA-Easytrieve/Plus is so smart, if you do not 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 (described in Chapter 5, “Activity Section-Processing and Logic”), then CA-Easytrieve/Plus uses the output from that SORT.)

Since our sample program only has one input file (PERSNL), the INPUT parameter on the JOB statement is completely optional. Without it, CA-Easytrieve/Plus looks for input and uses the first file (the only file) in our library section, PERSNL.

### **Naming a JOB Activity**

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

You 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 described 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, which we describe in the following pages of this tutorial.

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

### **A New Condition**

We talked a lot about a report program that simply extracts some data from a file and prints it out. Granted, CA-Easytrieve/Plus makes doing this very easy and automatic, but it can do so much more!

## If This Then That

Let us imagine your boss just came in and said that the report you are working on has to include net pay and deductions. Let us look again at the program we were working on so far.

```
FILE PERSNL FB(150 1800)
NAME    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 NAME EMP# GROSS
```

The program does access 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. It just so happens, you are a payroll expert and you know that employees who make \$500 or more get a 28 percent deduction; the rest do not get any deduction because they are too poor. (Or at least you think so because you are one of them!)

You can state the condition we just described 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, then deduct 28 percent to give the net pay. Otherwise, if the gross is less than 500, then there are no deductions and net-pay is the same as gross."

CA-Easytrieve/Plus requires an END-IF to complete the expression.

### Adding Logic to 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)
  NAME 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                    } new
  ELSE                                           } logic
    NET-PAY = GROSS                               }
    DEDUCTIONS = 0                               }
  END-IF                                         }

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

There are a couple of 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/Plus Working Storage

Unlike many other languages, CA-Easytrieve/Plus 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 that normally describes the start location with the letter W.

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

You can describe the above fields in words as working storage fields, four characters long, in packed decimal format with two decimal places. Let us place these fields in the library section of our program (this lets them be more easily seen than if they were placed in the activity section).

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

So far, we used some elementary logic, calculated some values, and created a place to store those values. There is one more thing we have to do to get those values printed on our report. We have to tell CA-Easytrieve/Plus where to print them.

## CA-Easytrieve/Plus LINE Statement

The last line in the program we showed you looks like this:

```
LINE 01 DEPT NAME EMP# GROSS
```

This line of code prints the detail lines on the report. It tells CA-Easytrieve/Plus what fields to print and the order in which to print them.

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

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

We just add the names of our two new fields in the 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 made.

```

FILE PERSNL FB(150 1800)
  NAME      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 NAME EMP# GROSS NET-PAY DEDUCTIONS
    
```

And here is some sample output from the program just shown.

As you can see, two new columns of information were added for NET-PAY and DEDUCTIONS:

11/18/88	PERSONNEL REPORT EXAMPLE-1				PAGE	1
DEPT	NAME	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	

## Review of Job Activities

In this lesson of the tutorial, you learned how to add conditional logic and calculations to your program to compute new information. You also learned how to store the new information and format it for printing.

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.
- LINE determines which fields are on your report and in what order.

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

## For More Information

If you want to add more detail to your understanding of the JOB activity section, turn to Chapter 5, “Activity Section – Processing and Logic.”

If you want to continue with the tutorial, continue to the next lesson.

## Lesson 3: Printing CA-Easytrieve/Plus Reports

The CA-Easytrieve/Plus PRINT statement activates report statements that result in a printed report.

As we describe in the next 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, and 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 are executed against a record of the PERSNL file, the PRINT statement tells CA-Easytrieve/Plus to execute the report definition statements.

In the above 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/Plus executes the first report in the job activity section.

Once the report statements execute, 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/Plus does the rest.

## Editing Your Report Output

In the last lesson of this tutorial, we added two new values to our report, NET-PAY and DEDUCTIONS. Both of them, with GROSS, are dollar values. Up until now, dollar values only printed as ordinary numbers with decimal places. We can edit these values so that 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 specify how non-alphanumeric data prints. (You cannot edit alphanumeric fields.)

To give an example, let us add edit masks to the three currency fields in our example program so that 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)
```

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

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

11/18/88		PERSONNEL REPORT EXAMPLE-1			PAGE	1
DEPT	NAME	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 high-order zeros are suppressed and each value has one dollar sign. Any all-zero values in the DEDUCTIONS column print as blanks.

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

1. Each digit in a field must be designated in the edit mask. Since a four-byte packed decimal field can contain seven digits, we must designate seven digits in the mask. This is done with \$\$\$\$999.
2. Dollar signs (\$) in the edit mask indicate that a dollar sign prints before 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
	0001234	\$12.34
	0000123	\$1.23
	0000012	\$0.12
		—————>

As the number of high-order zeros increase, the dollar sign automatically *floats* to the right.

3. 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.

4. Commas and decimal points print just as indicated. In the above example, you can see that commas are suppressed with high-order zeros for numbers less than 1000.
5. When you use the same mask 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 all over again. Remember, multiple parameters and subparameters are enclosed in parentheses.

6. To suppress all-zero values from printing (if appropriate), you 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 to illustrate its use.

## Field Headings

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

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

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

By placing each word in single quotes, you indicate that CA-Easytrieve/Plus should *stack* the heading, one word over the other.

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

11/18/88		PERSONNEL REPORT EXAMPLE-1			PAGE	1
DEPT	NAME	EMPLOYEE NUMBER	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	MCAHON	04234	\$386.40	\$386.40		

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

## Reviewing PRINT, MASK, and HEADING

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

- PRINT activates a report declaration resulting in a printed report.
- MASK lets you change the look of fields on your report.
- HEADING lets you customize column headings on your report.

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

```
FILE PERSNL FB(150 1800)
NAME      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 NAME EMP# GROSS NET-PAY DEDUCTIONS
```

### For More Information

If you want to learn more about the PRINT statement, you can turn now to Chapter 6, “Activity Section-Input and Output”.

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

## Lesson 4: Report Declarations

The example program we are describing 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 NAME EMP# GROSS NET-PAY DEDUCTIONS
```

In this lesson, we describe these three statements and add the following four statements to our program:

- SEQUENCE
- CONTROL
- SUM
- HEADING

## REPORT Statement

The REPORT statement must be the first statement in your report declaration. It tells CA-Easytrieve/Plus that a report is about to be described and also identifies 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 leave 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 type programs in as we move along and review output at your terminal, then 80 characters per line is appropriate (since you can only get 80 characters on your screen).

## Report Definition Statements

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

There are six report definition statements in CA-Easytrieve/Plus. 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/Plus user came up with a useful mnemonic device for remembering these statements and their order:

Sisters	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					

Though it sounds silly, it is effective. All of these statements are described briefly on the remaining pages of this lesson.

## SEQUENCE Statement

The SEQUENCE statement sorts your report on a specified key in ascending or descending order. Let us sequence our current report example on department in ascending order. That is, let us tell CA-Easytrieve/Plus 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 right after the REPORT statement:

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

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

When we run our program, here is what we get:

11/18/88		PERSONNEL REPORT EXAMPLE-1			PAGE	1
DEPT	NAME	EMPLOYEE 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:** The records are now in order by department number.

When you use SEQUENCE, you do not need to define any extra files or additional input/output commands in your program; CA-Easytrieve/Plus takes care of that for you.

## CONTROL Statement

The CONTROL statement creates a control break on a specified field (called the control field). It automatically *totals* all quantitative fields (fields with decimal positions) at the time of the control break and *grand totals* at the end of the report.

Now that we 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 need to 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 NAME EMP# GROSS
```

Now, at the end of each department (and end-of-report), we get our totals as shown below:

11/18/88		PERSONNEL REPORT EXAMPLE-1			PAGE	1
DEPT	NAME	EMPLOYEE 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		

## SUM Statement

Let us say that you 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 us 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 NAME EMP# GROSS NET-PAY DEDUCTIONS
```

Now, GROSS is the only field totaled:

11/18/88		PERSONNEL REPORT EXAMPLE-1			PAGE 1
DEPT	NAME	EMPLOYEE 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		

## TITLE Statement

The TITLE statement gives us the title of our report. We were 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 NAME 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. You can omit the title number when you have only one title; it simply defaults to 01. When you have more than one title (and `TITLE` statement), you must number them in ascending order.

You saw what this statement does on our example report, but we show you again in case you forgot. The `TITLE` statement, shown above, is responsible for the title shown on the following report:

11/02/88		PERSONNEL REPORT EXAMPLE-1			PAGE	1
DEPT	NAME	EMPLOYEE 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 automatically print on the same line. We show you how to override this in Chapter 7, “Activity Section-Reporting.”

## HEADING Statement

The `HEADING` statement, like the `HEADING` parameter of the `DEFINE` statement (described 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 us say we decided that the field name, `NAME` 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 `NAME`, followed by the new column heading:

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

To be consistent with our other heading, `EMPLOYEE NUMBER`, we 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 quotes, are treated the same as any other multiple parameters.

Here is how it prints:

11/18/88		PERSONNEL REPORT EXAMPLE-1			PAGE	1
DEPT	EMPLOYEE NAME	EMPLOYEE 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			

## LINE Statement

The last report definition statement is one you have seen, with TITLE, since the beginning of this chapter. We described it briefly in Lesson 2.

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 in which we want them printed:

```
REPORT PAY-RPT LINESIZE 80
  SEQUENCE DEPT
  CONTROL DEPT
  SUM GROSS
  TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
  HEADING NAME ('EMPLOYEE' 'NAME')
> LINE DEPT NAME 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/Plus has no idea what detail information you want printed on your report or the order in which 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/Plus Report. In fact, you now have all the information to write your own standard reports by customizing what you 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 prints control totals and grand totals for all quantitative fields.
- SUM overrides control totals and totals only specified fields.
- TITLE prints major report titles.
- HEADING prints customized column headings.
- LINE tells CA-Easytrieve/Plus what fields to put on detail lines and in what order.

### For More Information

If you want to add more detail to your understanding of CA-Easytrieve/Plus report declarations, turn to Chapter 7, “Activity Section—Reporting.”

The next two lessons in this tutorial introduce you to CA-Easytrieve/Plus macros and diagnostics. To go on to Lesson 5, continue below.

## Lesson 5: Making Your Job Easier with Macros

A CA-Easytrieve/Plus macro is simply a portion of a program that you store somewhere for repeated use. It could be a file definition that you want to use in more than one program without typing it more than once, or it could be a piece of program logic or a report declaration that you use often in different programs.

The main goal of a macro is to save you from duplicating your effort. In our example program, we could place all of the field definitions for the PERSNL file in an CA-Easytrieve/Plus macro. From there, many program could access them.

To do this, you store the code you want saved in the designated macro storage area at your site. The first line of any macro you save must consist of the word MACRO. The storage facility you have at your site also requires that you give the macro a name.

Once your macro is stored and has a name, you just call it into your program whenever you need it by specifying the name preceded by a percent symbol (%).

To substitute a macro for the CA-Easytrieve/Plus field definition statements in our sample program (assuming we named the macro PERSNL and stored it somewhere on the system), we just replace them with the statement %PERSNL:

```
FILE PERSNL FB(150 1800)
%PERSNL                                     } macro
NET-PAY   W  4  P  2  MASK A                } invocation
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
SEQUENCE DEPT
CONTROL DEPT
SUM GROSS
TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
HEADING NAME ('EMPLOYEE' 'NAME')
LINE DEPT NAME EMP# GROSS NET-PAY DEDUCTIONS
```

### For More Information

For more detailed information on invoking macros, turn Chapter 8, "Macros."

Otherwise, continue with Lesson 6.

## Lesson 6: Diagnosing An Error

CA-Easytrieve/Plus makes diagnosing most errors a very simple process. If you were following along and typing program code in during the course of this tutorial, it is possible you made some mistakes. A common error is to mistype a CA-Easytrieve/Plus keyword or the name of a field. If you try running a program with such an error, you are confronted with a program listing that looks something like this:

```

11/30/88 11.52.02      CA-Easytrieve/Plus - 6.0      PAGE 1
                        YOUR COMPANY NAME
PROGRAMS AND ALL SUPPORTING MATERIALS COPYRIGHT 1982 BY PSI
 1 FILE PERSNL FB(150 1800)
 2   NAME          17  8  A
 3   EMP#           9  5  N
 4   DEPT           98  3  N
 5   GROSS          94  4  P 2  MASK (A '$$, $$9.99')
 6   NET-PAY        W  4  P 2  MASK A
 7   DEDUCTIONS     W  4  P 2  MASK (A BWZ)
 8 JOB INPUT PERSNL NAME FIRST-PROGRAM
 9   IF GROSS GE 500
10     DEDUCTIONS = .28 * GROSS
11     NET-PAY = GROSS-DEDUCTIONS
12   ELSE
13     NET-PAY = GROSS
14     DEDUCTIONS = 0
15   END-IF
16   PRINT PAY-RPT
17 REPORT PAY-RPT LINESIZE 80
18   SEQUENCE DEP
18 *****B082 NAME IS UNDEFINED-DEP
19   CONTROL  DEPT
20   SUM      GROSS
21   TITLE 01 'PERSONNEL REPORT EXAMPLE-1'
22   HEADING NAME ('EMPLOYEE' 'NAME')
23   LINE 01 DEPT NAME EMP# GROSS NET-PAY DEDUCTIONS
OPTIONS FOR THIS RUN-ABEXIT SNAP  DEBUG (STATE FLDCHK NOXREF)
                        LIST (PARM FILE)  PRESIZE  512
                        SORT (DEVICE SYSDA  ALTSEQ NO MSG DEFAULT
                        MEMORY MAX WORK  3)  VFM ( 16)
*****A014 PREMATURE TERMINATION DUE TO PREVIOUS ERROR(S)

```

The mistake we made in this program (on line 18 of the listing) is that we mistyped the field name DEPT. We left off the T. The error message we received looks like this:

```
***** B082 NAME IS UNDEFINED-DEP
```

CA-Easytrieve/Plus is telling us that it does not know what a DEP is. This message consists of a message ID, a diagnostic message, and a supplemental message that identifies the mistyped field name (DEP).

Message ID	Diagnostic Message	Supplemental Message
*****B082	NAME IS UNDEFINED	DEP

To correct your error, you just retype the misspelled field name in your program. You can, of course, come across other error messages, all of which can be easily looked up in Appendix A of the *CA-Easytrieve/Plus Reference Guide*. There, you can find complete descriptions of all CA-Easytrieve/Plus diagnostic messages.

## Summing Things Up

In this tutorial, you followed the development of a moderately complex CA-Easytrieve/Plus report program. We started with a very simple report and built on it until we used all of the CA-Easytrieve/Plus basic report writing features.

You should now have a good understanding of the CA-Easytrieve/Plus program structure and how to use most of the basic report writing tools CA-Easytrieve/Plus 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 report writing process. The other parameters and statements are covered in other portions of this document or in the *CA-Easytrieve/Plus Reference Guide*.

### For More Information

If you want more information on CA-Easytrieve/Plus diagnostics, turn to Chapter 9, "Programming Techniques."

Or, read any first level readings you skipped.

Or, start the second reading in Chapter 4, "Library Section-Describing and Defining Data."

# Library Section-Describing and Defining Data

---

## Introduction

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

Before taking you into deeper descriptions of specific CA-Easytrieve/Plus statements, this chapter illustrates some general rules of syntax use.

In this chapter, you find:

### Reading 1

- CA-Easytrieve/Plus 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
- Using the EXIT parameter.

## Syntax Rules

Before we get started, we want to mention some of the general rules concerning CA-Easytrieve/Plus syntax.

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

### Statement Area

All CA-Easytrieve/Plus 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/Plus code anywhere in columns 1 through 72. You can indent or align certain statements for readability, but it is 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/Plus statement can start at the next available position of the statement area (after the space). For example, the following two CA-Easytrieve/Plus statements are on one line:

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

### Comments

When the first nonblank character of a statement is an asterisk (\*), the remainder of that line is considered to be a comment.

**Note:** The CA-Easytrieve/Plus compiler ignores it.

You can use comment statements at any place in a program, except in a continued statement. CA-Easytrieve/Plus treats a statement containing all blanks as a comment.

## Continuations

The last nonblank character of a statement terminates the statement unless that character is a hyphen (-) or a plus sign (+).

- A hyphen indicates that the statement continues at the start of the next statement area.
- A plus sign indicates that the statement continues with the first nonblank character in the next statement area.

The difference between minus and plus 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/Plus statement. A word can be a keyword, field name, literal, or symbol. All words begin with a nonblank 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/Plus delimiters are:

Delimiter	Description
space	The basic delimiter in each statement.
' apostrophe	Encloses literals that are alphabetic.
. period	Terminates a statement.
, comma	Used optionally for readability.
( ) parentheses	Encloses multiple parameters and portions of arithmetic expressions (the left parenthesis acts as a basic delimiter).
: colon	Used as a delimiter for file, record, and field qualifications.

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

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

## Keywords

Keywords are words that have specific meaning to CA-Easytrieve/Plus. Some keywords are reserved words. You can use nonreserved keywords in the appropriate context as field names whereas you cannot use reserved words as field names. See the *CA-Easytrieve/Plus Reference Guide* for a list of all reserved keywords.

## Multiple Parameters

You must enclose multiple parameters in parentheses to indicate group relationships. The following example is an CA-Easytrieve/Plus statement with multiple parameters:

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

## Field Names

Field names are composed of a combination of not more than 40 characters chosen from the following:

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

The first character of a field name must be an alphabetic character or a decimal digit. Also, 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 and all field names in a single file must be unique. 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.

You can qualify the field name RECORD-COUNT in the following ways:

```
FILEA: RECORD-COUNT  
FILEA:RECORD-COUNT  
WORK : RECORD-COUNT  
WORK:RECORD-COUNT
```

## Labels

Labels identify specific JOBS, PROCedures, REPORTs, and statements. Labels can be 40 characters long, can contain any character other than a delimiter, and can begin with A-Z or 0-9. They cannot consist of all numeric characters.

## Alphabetic Literals

Alphabetic literals are words that are meant to be taken literally. They are enclosed in apostrophes, and can be 254 characters long. They can only contain EBCDIC characters. Whenever an alphabetic literal contains an embedded apostrophe, you must code two apostrophes.

For example, code the literal O'KELLY as:

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

## Numeric Literals

Numeric literals can contain 18 numeric digits (EBCDIC 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 EBCDIC values that contain characters not available on standard data entry keyboards. Prefix an EBCDIC hexadecimal literal with the letter X and an apostrophe (X'), and terminate it with an apostrophe.

CA-Easytrieve/Plus compresses each pair of digits that you code in the apostrophes into one character. CA-Easytrieve/Plus permits only the EBCDIC digits 0 to 9 and the letters A to F. The following hexadecimal literal defines two bytes of binary zeroes:

```
X'0000'
```

## Identifiers

Identifiers are words that name things, such as field name, and statement labels in CA-Easytrieve/Plus. Identifiers cannot contain these delimiters.

- , comma
- ' apostrophe
- ( left parenthesis
- ) right parenthesis
- : colon

## Arithmetic Operators

CA-Easytrieve/Plus arithmetic expressions (see Chapter 5, "Activity Section – Processing and Logic") use the following arithmetic operators:

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

The arithmetic operator must lie between two spaces.

## Describing Files and Fields

You must describe all of the files and their associated fields, including working storage fields, referenced in your CA-Easytrieve/Plus program before you reference them.

## Defining Data

Normally, you define data fields in the chapter 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:

```

                                ADDRESS FIELD
                                -----
RECORD { Jones, John J.  16822 Evergreen  Chicago ... }
          Hammond, Martha 422 Ash Ave.   Evanston .. }
          Gray, Frederick 16 Apple St.  Lockport .. }
          Freud, William G. 754 Lake St.  Peotone .. }
          _____
          _____
          _____
          .
          .
          .
}
```

F  
I  
L  
E

## Defining File Attributes

The FILE statement describes 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. You can define two categories of data:

- File data (fields defined in 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 statements at the beginning of the library section.

The general structure of the FILE statement is:

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

### FILE

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

### file-name

File-name is a one-to eight-character name (one-to seven in VSE) that defines your file to CA-Easytrieve/Plus. All input/output statements that operate on the file refer to this name. File-name is also 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 in your program.

### file attributes

The CA-Easytrieve/Plus FILE statement has a host of parameters that 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, depending on your particular operating environment. See the *CA-Easytrieve/Plus Reference Guide* or *Pocket Reference* for the complete FILE statement syntax.



## field-length

You specify the length of a field in bytes (characters and 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 five data-types.

Type	Maximum Field Length (in bytes)
A Alphanumeric	32,767
N Numeric	18
P Packed	10
U Unsign Packed	9
B Binary	4

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

```
NAME 17 8 A
```

## decimal-positions

By specifying *decimal-positions* in your field description, you:

- Identify a field to CA-Easytrieve/Plus as quantitative, that is, 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.
- Enable proper placement of commas and decimals with leading (high-order) zeros suppressed when the field is printed.

Four types of data can have decimal positions:

```
N (Numeric)
P (Packed)
B (Binary)
U (Unsigned Packed)
```

Specify the decimal-positions by coding an integer (0 through 18) after the data-type. For example, the following is a five-byte numeric field with two decimal positions:

```
AMOUNT 40 5 N 2
```

## HEADING Parameter

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 in single quotation marks. For example,

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

produces the column heading:

```
CLIENT NAME
```

To *stack* a column heading so that each word appears on top of each other, 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 Parameter

The MASK parameter creates a customized edit mask. An edit mask is an optional pattern of characters specifying how non-alphanumeric data prints. (You cannot edit alphanumeric fields.) An edit mask is created using combinations of the following characters:

Character	Description
9	Formats digits.
Z	Suppresses leading zeros.
*	Replaces leading zeros with an asterisk.
-	Prints a minus sign before the first nonzero digit of a negative number.
\$	Prints a currency symbol before the first nonzero 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

You can include commas 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 can 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

As shown earlier, the general structure for an edit mask is:

```
[MASK { [mask-name] [BWZ] ['edit-mask']}]
```

- MASK is the CA-Easytrieve/Plus keyword, indicating an edit mask follows.
- The mask-name names 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 define a mask, you do not have to define it again to use it again.
- BWZ (blank when zero) specifies that a field should not print 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-name.
- The 'edit-mask' 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.

<b>MASK</b>	<b>RESULT</b>
'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

You can mask fields that have the potential for containing a negative value in such a way that an indicator of their negativity displays when printed. An indicator of negativity, such as minus sign (-) or the letters CR (for credit), or any other chosen indicator, only print when the field contains a negative value.

To do this, you mask the field as you would normally, making sure all digits are accounted for, then you 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.

<b>MASK</b>	<b>RESULT</b>
'\$\$\$,\$\$\$,999 CREDIT'	\$12,345,678 CREDIT
'\$\$\$,\$\$\$,999-'	\$12,345,678-
'Z99,999,999-'	12,345,678-

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

### Default Edit Masks

Fields that are defined with positions to the right of a decimal point are known as *quantitative*. These fields have system default edit masks that account for the automatic printing of commas and decimal points in printed totals. Numeric fields without defined decimal positions print without commas or decimal points and are not automatically totaled 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 while a CA-Easytrieve/Plus program runs.

Define working storage by specifying W as the start-location. For example, the following defines a numeric working storage field four characters long with two decimal positions.

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

This field could be defined in the library section or in an activity before being referenced.

### DEFINE within an Activity

You usually specify file fields and working storage fields in your CA-Easytrieve/Plus library section, but you can also define them in 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 keyword DEFINE 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.

```

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

Activities   { JOB INPUT PERSNL NAME MYPROG
              { EMP-COUNT = EMP-COUNT + 1
              { PRINT REPORT1
...          { *
              { REPORT REPORT1
              { LINE EMP# NAME EMP-COUNT

```

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

```

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

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

```

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

#### END OF FIRST READING

<b>If</b>	<b>Then continue with. . .</b>
You completed the tutorial	Chapter 5, "Activity Section-Processing and Logic"
You branched to this chapter from the tutorial	Lesson 2 in Chapter 3, "Standard Reporting with CA-Easytrieve/Plus"

## YOU ARE NOW STARTING THE SECOND READING OF CHAPTER 4

### Defining Static Working Storage

Static Working Storage fields are fields used for storing accumulated values that print at the end of a report or are used to compute some other values at the end of a report or at control breaks (such as averages).

You define Static Working Storage fields 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/Plus processes reports. In the first reading of Chapter 5, "Activity Section-Input and Output," we describe the PRINT statement and the process that occurs when reports are sequenced (through the SEQUENCE statement) or are multiple in one JOB activity (more than one REPORT statement is used).

In both cases, CA-Easytrieve/Plus outputs data to an intermediary file called a work file or spool file. (See the examples in Chapter 6.) Work files do not get formatted into reports (through report definition statements) until they are first sequenced or until the system printer becomes available.

It is due to the use of intermediary work files that a need for two different types of working storage fields arises. To understand the need for these two field types and the differences between them, the following description provides a contrast that sheds light on the use of both.

#### 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 accumulates 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 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 that are in error. This is only true for non-static working storage fields (type W) that accumulate values for sequenced reports. For example:

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

In the above example, the W-type field increments by one (1) each time a record is processed. Once the work file is 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 before sequencing. If you tried 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			Work File Before SEQUENCE	
SEQUENCE KEY FIELD	S-TYPE ACCUMULATOR FIELD		SEQUENCE KEY FIELD	S-FIELD VALUES NOT PASSED TO WORK FILE
ZZZ	1	→	ZZZ	
BBB	2		BBB	
AAA	3		AAA	
PPP	4		PPP	
QQQ	5		QQQ	
SSS	6		SSS	
CCC	7		CCC	

The above example illustrates the fact 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, the VALUE parameter below assigns an initial value of JANUARY to the alphanumeric working storage field CURR-MON.

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

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

#### RESET Option

The RESET option is used only for W working storage fields. When coded on the field definition for a W field, RESET returns the field to its initial value whenever JOB or SORT executes. You cannot use RESET for redefine fields (fields having overlay redefinition).

### 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, could have been originally entered 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/Plus, 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
| L L L L L L 0 2 1 0 5 5 L L L L L L |
|         |   |   |   |   |         |
position  103 105 107
in
record:

```

In this example, 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 that 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 above example).

When using overlay redefinition, make sure that the redefining fields fits in 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, the following defines contiguous fields in a record.

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

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.

END OF SECOND READING

Please continue with Chapter 5, "Activity Section-Processing and Logic."

YOU ARE NOW STARTING THE THIRD READING OF CHAPTER 4

## FILE Statement Revisited

In the first reading of this section, we described the FILE statement and said that it was necessary for describing input and output files. Also, we said that there are a number of FILE statement parameters. In this reading, we describe two FILE statement parameters that you might find very useful.

## Virtual File Manager (VFM)

The VIRTUAL parameter of the FILE statement invokes the virtual file management facility (VFM) of CA-Easytrieve/Plus. The parameter has this structure:

```
FILE file-name {VIRTUAL [RETAIN]} { F }
               { V }
               { U } logical-record-length
```

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/Plus keywords.

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

VIRTUAL

The *VIRTUAL* parameter designates that the named file is 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 remains in memory until the end of the associated CA-Easytrieve/Plus execution. If *RETAIN* is not specified, the VFM file is deleted once it is read back into your program.

**Record Length and Type**

CA-Easytrieve/Plus 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 nonstandard data files CA-Easytrieve/Plus does not process directly. *EXIT* is not valid for VFM, IMS/DLI, or CA-IDMS.

Always used on the *FILE* statement, it has this structure:

```
FILE file-name [EXIT (program-name [NR] +
[ { field-name }... ) ] [MODIFY] )]
[ { 'literal' } ]
[ { } ]
```

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

**NR**

*NR* has meaning only in VSE. *NR* indicates that the program is non-relocatable.

**USING**

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

**MODIFY**

*MODIFY* specifies that CA-Easytrieve/Plus provides input or output services, yet *EXIT* can inspect and modify each record after input and before output.

## 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. The COPY statement looks like this:

```
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/Plus cannot uniquely identify the data reference. You can qualify fields in CA-Easytrieve/Plus by preceding them with their file name and a colon. For example, OUTFILE:NAME.

### Example of COPY Statement

```
FILE PERSNL FB(150 1800)
  NAME          17  20 A    HEADING ('EMPLOYEE NAME')
  NAME-LAST NAME    8 A    HEADING ('FIRST' 'NAME')
  NAME-FIRST NAME +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
```

END OF THIRD READING

Please continue with Chapter 5, "Activity Section-Processing and Logic."



# Activity Section-Processing and Logic

---

## Introduction

The activity section of a CA-Easytrieve/Plus program is where all processing logic and report declarations reside. You might say it is where the action is. The activity section can contain two types of activity:

- SORT activity
- JOB activity

SORT activities are signified by the SORT statement and JOB activities by the JOB statement.

In this chapter, you find:

### Reading 1

- CA-Easytrieve/Plus 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 and DO WHILE
- STOP statement

### Reading 3

- Sorting data and the SORT statement
- Processing with procedures including START and FINISH procedures
- Processing tables.

## JOB Activities

### JOB Statement

The JOB statement defines and initiates processing activity. Also, it identifies the name of the automatic input file. In its basic form, it appears like this:

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

You can code JOB statement parameters in any order.

#### INPUT

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

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

#### file-name

File-name identifies the automatic input files. 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 used only for documentation purposes. The job-name can be up to 40 characters long, can contain any character other than a delimiter, and begin with A-Z or 0-9. It cannot consist of all numeric characters.

This example shows the location of the JOB statement and the subactivities in a CA-Easytrieve/Plus program.

```
          ** Library **
          *
ACTIVITY> JOB INPUT PERSNL NAME EXAMPLE
LOGIC >  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 NAME 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 appropriate report.

## Conditional Expressions

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

### IF Statement

Processing in a JOB activity can depend 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 statements generally have the following format:

```

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

      [statements executed for true 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 NAME = '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

CA-Easytrieve/Plus permits the following arithmetic operators in conditional statements:

Operators	Meaning
EQ =	Equal to
NE $\neq$	Not equal to
GT >	Greater than
GE $\geq$	Greater than or equal to
LT <	Less than
LE $\leq$	Less than or equal to

**IF/ELSE**

ELSE directs CA-Easytrieve/Plus 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."

## Special IF Statements

Use special IF statements to check the integrity of the data in your files. A special IF statement has the following format:

```

                                { ALPHABETIC }
                                { NUMERIC   }
                                { SPACE     }
IF field-name [NOT]             { 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.
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
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 this statement, CA-Easytrieve/Plus 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.

This table helps you visualize the concept of logical connectors.

<b>When DIVISION = A, AMOUNT = 15, and STATE = GA</b>	
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
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 can alter the normal order of evaluation.

## Calculations

There are four arithmetic operations in CA-Easytrieve/Plus:

```
* 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. Calculations follow this format:

```
field-name { } { * } value-2
           { = } value-1 { / }
           { EQ}         { + }
           { }           { - }
```

### Parentheses in Calculations

You can use parentheses 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. The 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 then multiplied by 1.3 and the product of this is assigned to the RESULT field.

## END OF FIRST READING

<b>If ...</b>	<b>Then continue with ...</b>
You completed the tutorial	Chapter 6, "Activity Section-Input and Output"
You branched to this chapter from the tutorial	Lesson 3 of Chapter 3, "Standard Reports with CA-Easytrieve/Plus."

YOU ARE NOW STARTING THE SECOND READING OF CHAPTER 5.

## 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 data, rounding, or integerizing. Assignments generally follow this format:

```
field-name-1 [INTEGER] [ROUNDED ] { } {
                [TRUNCATED] {EQ} {field-name-2 }
                [          ] { } {literal }
                [          ] { } {arithmetic expression }
```

### Simple Assignment Examples

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

### ROUNDED and INTEGER

You can round fractional numbers or drop off the fractional component of a number by using the ROUNDED and INTEGER options of the assignment statement.

Use the INTEGER option after field-name-1 to ignore the fractional portion of the value being assigned. INTEGER transfers only the digits to the left of the decimal point during the assignment.

Use the **ROUNDED** option to round numbers that have a fractional component with more digits than the receiving field permits. Excess digits are truncated (on the right) once rounding is complete. Rounding follows these rules:

- Truncation is determined by the field length and decimal places of the result field.
- The least significant digit in the result field is increased by 1 if the most significant digit of the excess digits is greater than or equal to 5.
- If the **ROUNDED** option is not specified, truncation occurs without rounding (after decimal alignment). This is the same as specifying **TRUNCATED**, which is the default.
- Rounding of a computed negative result occurs by rounding the absolute value of the computed result and making the final result negative.

Using both **ROUNDED** and **INTEGER** rounds the result of an assignment to the nearest integer before dropping the fractional component.

#### Rounding Examples

Assuming that a program contains the following field definitions:

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

Then:

<b>Assignment Statement</b>	<b>RCVFLD Result</b>
RCVFLD INTEGER ROUNDED = SENDFLD	11.0
RCVFLD INTEGER TRUNCATED = SENDFLD	10.0
RCVFLD INTEGER = SENDFLD	10.0
RCVFLD ROUNDED = SENDFLD	10.8
RCVFLD TRUNCATED = SENDFLD	10.7
RCVFLD = SENDFLD	10.7

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

## 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

```
MOVE {file-name-1 } [send-length] TO {file-name-2 } +
    {field-name-1}      {field-name-2}
    {literal   }      {
[receive-length]      [FILL literal]
```

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 lets you place specified characters in the unused spaces of the new field (the default is blank spaces).

#### MOVE Example 1

```
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) is filled with asterisks.

### MOVE Format 2

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

You can use Format 2 to initialize the receiving field.

## MOVE Example 2

```
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. The general format of the MOVE LIKE statement is:

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

## MOVE LIKE Example

```

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

```

In the above example, the NAME field of INFILE1 is moved to the NAME 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 WHILE/END-DO Statements**

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

## Syntax

Its syntax is:

```
DO WHILE conditional-expression
  ** Logic **
END-DO
```

- The logic between DO WHILE and END-DO is executed until the conditional expression on the DO WHILE statement is no longer true.
- Conditional expressions follow the rules of IF 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 repeats “CTR = CTR +1” until CTR is equal to 10. At that point, control is transferred to the first statement after the END-DO statement.

### DO WHILE Nesting Example

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

```
JOB INPUT PAYROLL NAME DO-EX-2
          CTR1 = 0
        [ DO WHILE CTR1 LT 10
          [ CTR2 = 0
O [ I [ DO WHILE CTR2 LT 5
U [ N [ CTR2 = CTR2 + 1
T [ N [
E [ E [ ** Logic **
R [ R [
          [ END-DO
L [ CTR1 = CTR1 + 1
O [
O [ ** Logic **
P [
          [ 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 passes to the first statement following the outer END-DO statement.

### GOTO Statement

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

#### Syntax

```
{ GOTO } { label }
{      } {      }
{ GO TO } { JOB   }
```

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

GOTO Label

Label refers to a statement label. A statement label can be up to 40 characters long. The first character must be alphabetic. 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.

GOTO JOB

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

#### GOTO 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
      ** Logic **
      CHECK-REG-ROUTINE <-----
      ** More Logic **

```

## STOP Statement

A STOP statement lets you terminate an activity.

#### Syntax

Its format is:

```
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/Plus execution.

#### STOP Example

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

END of SECOND reading.

Please continue with Chapter 6, "Activity Section-Input and Output".

YOU ARE NOW STARTING THE THIRD READING OF CHAPTER 5.

## **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 permits. (The SORT activity uses the sort utility provided by your system.)

Syntax

The general format of the SORT statement is as follows:

```
SORT file-name-1 TO file-name-2 +  
    USING (field-name [D] ...) +  
    NAME sort-name
```

file-name-1

*File-name-1* is the input file to sort.

file-name-2

*File-name-2* is the output file.

USING field-name

*USING field-name* identifies those fields from file-name-1 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 is the major sort key, location is minor, and department is more minor.

D

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

NAME sort-name

NAME sort-name (like NAME on the JOB statement) identifies the sort activity and is used for documentation purposes only.

## Sort Example

```

FILE PERSNL FB(150 1800)
  NAME      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/Plus 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 appropriate to sort only certain records and to modify the contents. To do this, you must write a sort procedure that must immediately follow the SORT statement. A sort procedure is executed through the BEFORE parameter of the SORT statement:

```

SORT file-name-1 TO file-name-2 +
      USING (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 file-name-1 before passing the record to the sort.

## BEFORE proc-name

BEFORE proc-name identifies the user-defined procedure you want to execute.

CA-Easytrieve/Plus supplies input records to your sort procedure one at a time. If a BEFORE procedure is used, the SELECT statement must execute for each record that you want to sort.

- When you use BEFORE, 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.

The format of the SELECT statement is:

```
SELECT
```

SORT Procedure Example

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

```
FILE PERSNL FB(150 1800)
  NAME          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 selects only those records with a gross pay of greater than or equal to 500 for sorting.

## User Procedures (PROCs)

A user procedure (also called PROC for short) is a group of user-written CA-Easytrieve/Plus statements designed to accomplish some task. PROCs are useful when developing structured programs that modularize discrete and repetitive tasks. PROCs are invoked by using the PERFORM statement, which has the following format:

Syntax

```
PERFORM proc-name
```

proc-name

The *proc-name* can be one-to 40-characters long and must begin with a letter. It specifies the name of a user-defined procedure located at the end of the activity in which it is PERFORMed.

As mentioned earlier, procedures are discrete modules of program code that perform a task. When coded, they must have this format:

```
proc-name. PROC
** Procedure Logic **
END-PROC
```

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* that 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 in a procedure can invoke another procedure.

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 proc-name] [FINISH proc-name]
```

### START Procedure

START procedures execute routines before execution of the logic in the body of the JOB activity.

- The procedure is invoked automatically after the file is opened but before 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 6).

### FINISH Procedure

FINISH procedures identify a procedure to execute 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/Plus 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

### 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 recommended in a table. This recommendation is made to eliminate the need of performing a binary search, which is our method of accessing 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/Plus FILE statement:

```
FILE file-name TABLE { INSTREAM }
                      {          }
                      { literal }
```

### TABLE

The *TABLE* parameter of the FILE statement declares that the file is the object of a CA-Easytrieve/Plus SEARCH statement that accesses tables. Tables can be either:

- External – stored in a file outside your program

or

- Instream – data included in your program.

External table files must be sequentially accessible.

### INSTREAM

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

### literal

Literal specifies the number of entries (records) in an external table. Specify a value here only if the number of entries is greater than 256.

### 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
```

This example defines a table of state names that can now be looked up according to a two-digit code.

## Accessing Table Files

### SEARCH Statement

The SEARCH statement performs a search of a table. SEARCH can be:

- Coded any place in a JOB activity
- Issued any number of times against any number of tables.

### Syntax

The search statement has this format:

```
SEARCH file-name WITH field-name-1 GIVING field-name-2
```

### file-name

*File-name* is the name of the table that appears on the FILE statement.

### field-name-1

*Field-name-1* is the name of a field that contains a value that is compared to the search argument. It must be the same length and type as the search argument (ARG).

### field-name-2

*Field-name-2* is the name of a field into which the description is placed if a match exists between field-name-1 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 field-name-1 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
NAME          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 NAME GROSS-PAY ZIP POST-OFFICE-DESC
```

END OF THIRD READING

Please turn Chapter 6, "Activity Section-Input and Output".

# Activity Section-Input and Output

---

## Introduction

In CA-Easytrieve/Plus, file input/output can be controlled by CA-Easytrieve/Plus (automatic) or 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 find:

### Reading 1

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

### Reading 2

- User Controlled Input/Output of sequential access files including use of:

DISPLAY  
GET  
PUT

### Reading 3

- Use of the POINT statement to establish a starting position for sequential processing of a keyed file
- Programmer Controlled Input/Output of randomly accessed files including use of:

READ  
WRITE

## Automatic Input and Output

CA-Easytrieve/Plus 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 or files for automatic input to the JOB activity. All you have to do is specify the file name after the word INPUT. CA-Easytrieve/Plus takes care of all the rest. 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 do not see in your program. Here are the steps actually taken when the JOB statement executes 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 JOB OR SORT ACTIVITY
    ELSE
      PERFORM LOGIC ACTIVITIES
    END-IF
      RESET WORKING STORAGE
----- GOTO
```

You can leave the INPUT parameter off the JOB statement. If you do, CA-Easytrieve/Plus 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

Report printing is initiated through the CA-Easytrieve/Plus PRINT statement, which looks like this:

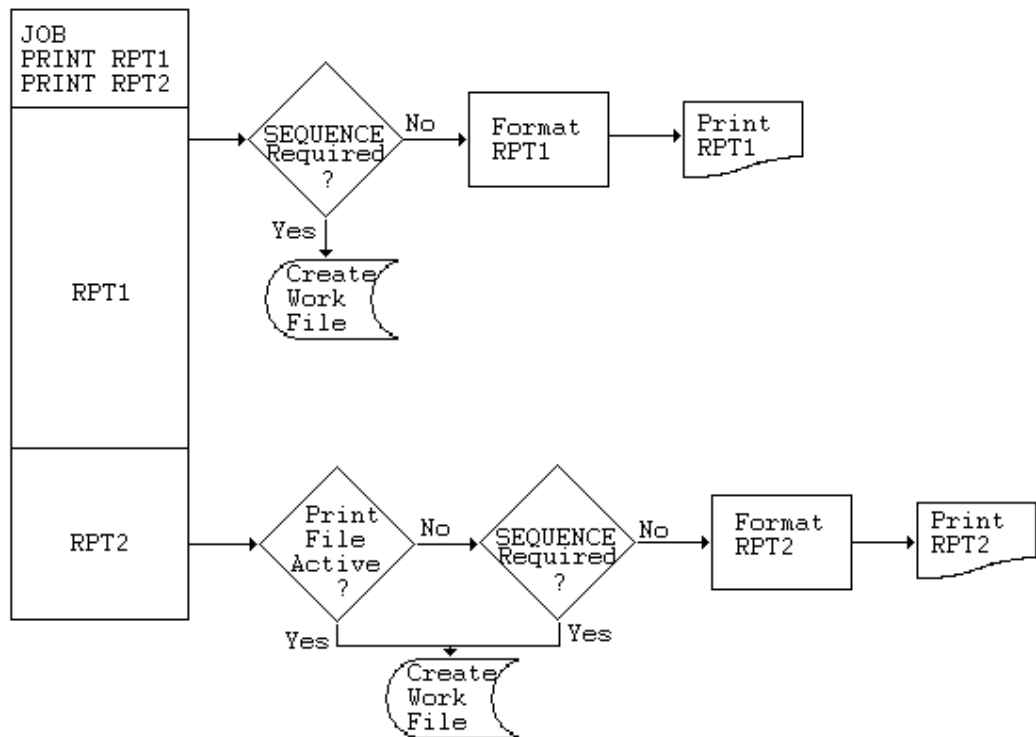
```
PRINT [report-name]
```

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.

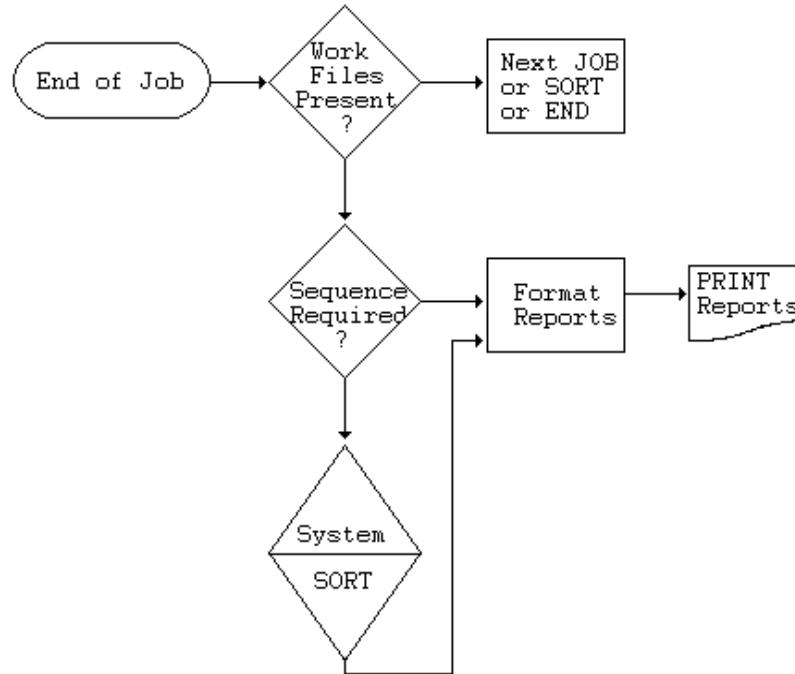
In most cases, however, your reports do not go directly to a printer (through a print file). Rather, they go to an CA-Easytrieve/Plus work file (sometimes called a spool file). Work files are necessary in two cases:

- 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 (that is, a SEQUENCE statement is present).

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



If work files are created, as presented in the previous graphic, they are held until the end of the job activity. At end of job, they are processed as follows:



END OF FIRST READING

<b>If ...</b>	<b>Then continue with ...</b>
You completed the tutorial	Chapter 7, "Activity Section-Reporting"
You branched to this section from the tutorial	Lesson 4 or Chapter 3, "Standard Reports with CA-Easytrieve/Plus"

---

## User Controlled Input and Output

YOU ARE NOW STARTING THE SECOND READING OF CHAPTER 6.

### Sequential File Processing

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

STATEMENT	PURPOSE
DISPLAY	Normally, displays data to your system output device.
GET	Sequential retrieval of input file records.
PUT	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.
- Highlighting reports.
- 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 DISPLAY are interspersed throughout the report (every other line) in an unSEQUENCED report or printed at the beginning of a SEQUENCED report (before the first title). When you use DISPLAY in report procedures, you are only permitted to display to the system output device (not to a data file).

The DISPLAY statement has three different formats. The next two pages show two of the formats with explanations and examples of the use of each. The third format is used only with extended reporting and is explained in the *CA-Easytrieve/Plus Extended Reporting Facility Guide*.

## DISPLAY Format 1

```
DISPLAY [file-name] [NEWPAGE] [ [ ] integer ] +  
                  [SKIP number] [ [ ] integer ]  
                  [ [ ] integer ]  
                  [ COL integer ]  
                  [ POS integer ]  
                  [ [ ] integer ]  
                  [ [ ] integer ]  
[literal-1] [literal-n]  
[field-name-1] ... [field-name-n]  
[ [ ] integer ]
```

file-name

When you specify *file-name*, CA-Easytrieve/Plus prints data to the named file. If you do not specify file-name, the default is SYSPRINT/SYSLST (the system output printers for OS/390 and VSE, respectively).

NEWPAGE

The NEWPAGE option specifies that a skip to a new page occurs before the data is printed.

SKIP number

The *SKIP number* option specifies that the designated number of lines (number) are skipped before the data is printed.

Integer

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

COL integer

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

POS integer

When used on report procedures, the *POS integer* option positions the next display item under the corresponding position on the LINE 01 statement.

literals or field-names

Code *literals or field-names* in the order you want them to appear on the printed line.

Examples of Format 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

```

DISPLAY [file-name] [ [NEWPAGE ] ] HEX [field-name]
                  [SKIP number] [file-name]
                  [ ] [ ]

```

In this format, CA-Easytrieve/Plus 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.

Example of Format 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. Its format is:

```
GET file-name
```

file-name

*File-name* identifies the input file defined in the library section.

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

GET Example

```

FILE MASTER FB(150 1800)
EMP#   9   5   N
NAME  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# NAME GROSS

```

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

```
JOB INPUT NULL
```

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

## PUT Statement

The PUT statement outputs to a file sequentially. Its format is:

Syntax

```
PUT  outfile  [FROM file-name]
```

outfile

*Outfile* identifies a file defined in the library section to which you are outputting data.

FROM file-name

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

PUT Example 1

```
FILE PERSNL FB(150 1800)
  EMP#      9  5  N
  NAME     17 16  A
  GROSS    94  4  P 2
FILE NEWPAY2 F(20) VIRTUAL RETAIN
  NAME      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
  NAME     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
```

END OF SECOND READING

Please continue with Chapter 7, "Activity Section-Reporting."

YOU ARE NOW STARTING THE THIRD READING OF CHAPTER 6.

## POINT Statement

The POINT statement establishes a starting position for sequential processing of a keyed file. This statement is for use on ISAM and VSAM files.

CA-Easytrieve/Plus 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 by automatic file input or a GET statement.

### Syntax

The format of the POINT statement is:

```
POINT file-name { EQ } {          }
                { =  } { field-name } [STATUS]
                {   } {          }
                { GE } { literal   }
                { >= } {          }
```

### file-name

*File-name* is an indexed, keyed, or relative-record file described on a FILE statement in the library section of your program.

### field-name or literal

You can use any valid *field-name or literal* 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

*STATUS* sets the system defined field FILE-STATUS with a return code from the operating systems data management control program. 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 should always contain a value of zero after a successful I/O request. This parameter is also used on the READ and WRITE statements.

## POINT Example

The following example causes sequential processing of a VSAM 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 VS
EMP#  9  5  N
NAME 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 keyed VSAM and to relative-record VSAM files.

#### Syntax

Its format is:

```
READ file-name KEY {field-name}
                  {'literal'} STATUS
                  {
```

file-name

*File-name* is the VSAM file-name located on a FILE statement in the library section of your program.

field-name/literal

*Field-name* contains the value of the VSAM key to find. You can also express this key value as a *literal*.

## READ Example: Programmer Controlled Random Access

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 that 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 DISPLAY an error message.

```

FILE PAYROLL
  EMP-NO  1  3  N
FILE MASTER VS
  EMP-NAME 40 10 A
*
JOB INPUT PAYROLL NAME READ-EXAMPLE
-----> READ MASTER KEY EMP-NO STATUS
        IF MASTER:FILE-STATUS NOT ZERO
            DISPLAY ERRRPT '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 a VSAM master 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.

## Format 1

Use Format 1 when adding to or updating a VSAM record.

```

WRITE file-name-1 [UPDATE] [FROM file-name-2 ] STATUS
                  [ADD   ] [
                  [

```

## WRITE Example

The example on the following page 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 VSAM file that 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  VS (UPDATE)
  GROSS  15  4  P  2
*
JOB INPUT TRANS NAME UPDATE-PGM
READ PAYVS KEY EMP-NO STATUS
IF PAYVS:FILE-STATUS = 16 . * 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

```

Format 2

Use Format 2 for deleting a VSAM record.

```
WRITE file-name-1 DELETE STATUS
```

Example

```

FILE TRANS
  TRANS-KEY  14  3  A
  TRANS-CODE 17  1  A. * TRANS-CODE value of D means Delete
*
FILE PAYVS VS (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

```

END OF THIRD READING

Please continue by reading Chapter 7, "Activity Section-Reporting."

# Activity Section-Reporting

---

## Introduction

One of the most powerful features of CA-Easytrieve/Plus is the easy-to-use reporting facility. Seven basic statements control most aspects of report production:

- REPORT
- SEQUENCE
- CONTROL
- SUM
- TITLE
- HEADING
- LINE.

Procedures are also available for customization of reports permitting great flexibility and user control.

In this chapter you 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.

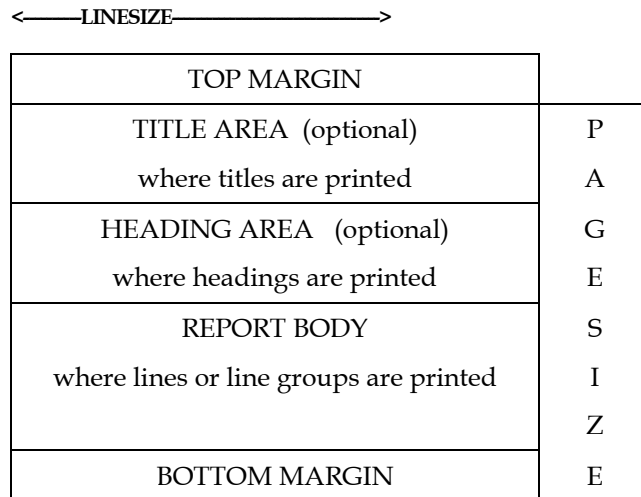
## Standard Reports

The report facility of CA-Easytrieve/Plus includes all of the functions necessary to produce most reports very easily. Using CA-Easytrieve/Plus report options, you can produce almost any report format. Most reports, however, are variations of what is termed the standard report.

Typically, standard reports consist of the following items:

- Titles
- Headings
- Lines or line groups.

The following example shows the basic structure of a standard report.



We describe how CA-Easytrieve/Plus formats and generates titles, headings, and lines on the following pages.

## Titles

The title is the first item printed on each report page. The TITLE statement in your program specifies the report title. You can have up to 99 TITLE statements in your program. The following example shows the title area of a report.

```

<-----LINESIZE----->
      SYSDATE                      PAGEWRD   Page
      |                               |       count
      V                               V         V
    99/99/99                      PAGE  ZZ,ZZ9
                                     TITLE 01  items
                                     TITLE 02  items
                                     ...
                                     TITLE 04  items
                                     ...
                                     TITLE 99  items
  
```

When more than one TITLE statement is coded in your program, they must be followed by sequence numbers (01 - 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 in 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 example shows two TITLE statements and their 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
  
```

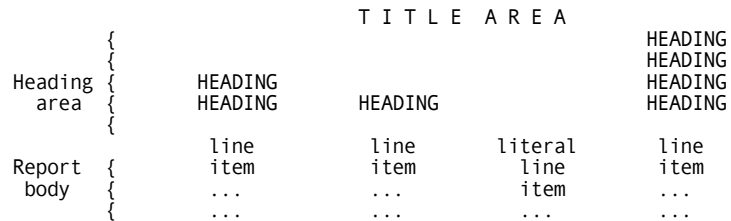
**Note:** A blank line was inserted where a TITLE 02 item would have otherwise printed.

## 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/Plus uses the field names of the DEFINE statement as headings.
- You can specify headings with 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 that 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 example shows the positioning of headings in a typical report:



This next example shows how you can define headings in your program.

Statements:

```

FILE PERSNL FB(150 1800)
> SSN      4 5 P HEADING('SOCIAL' 'SECURITY' 'NUMBER')
   NAME    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 NAME SSN '* NO OVERTIME *' PAY-NET
    
```

Produce:

	NAME	SOCIAL SECURITY NUMBER	* NO OVERTIME *	NET PAY
	WIMM 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 ...}
...
```

## Line Item Positioning

Line item positioning follows three rules:

1. LINE 01 items and their associated headings are centered in an area whose length is automatically controlled by the longer of the following:
  - The line item
  - The longest heading entry

The resulting value is called the item length.

2. The first line item in any LINE statement after LINE 01 (that is, LINE 02 through LINE 99) is positioned under the first line item of LINE 01. The data is left-justified under the LINE 01 data, regardless of the heading size.
3. Blank characters (spaces) separate all line items according to the value of the SPACE parameter (if any) of the REPORT statement. The SPACE parameter overrides automatic spacing.

When CA-Easytrieve/Plus analyzes a LINE statement according to the above rules, the total number of characters on that line must not exceed the value specified on the LINESIZE parameter of the REPORT statement.

The following example illustrates line item positioning:

```

FILE PERSNL FB(150 1800)
  SSN          4 5 P  MASK '999-99-9999' +
                HEADING('SOCIAL' 'SECURITY' 'NUMBER')
  NAME         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 NAME SSN SEX
LINE 02 ADDR-STREET ADDR-CITY

      item area      item area      item area
      1 5 10 15 1 5 10 1 4
      .....

      EMPLOYEE NAME      SOCIAL
                        SECURITY
                        NUMBER      SEX } heading
                        CODE}

      WIMN GLORIA      025-30-5228 1 } line
      430 M ST SW 107  BOSTON      } group
                        }
  
```

## Report Processing

The PRINT statement described in Chapter 6 identifies records for output and initiates the execution of a report declaration. It does not directly cause the printing of the report. Printing and formatting of a report are 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:

- 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 let you 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/Plus-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 describe 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 print positions).

SKIP

Number of blank lines to insert 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 before first heading or detail line (default is 3).

SPREAD

Spreads the columns of data evenly over the entire line, overrides the SPACE parameter (default is NOSPREAD).

NOADJUST

Left-justifies the title lines and report on the page. The default is centering the report on the page. SPREAD and NOADJUST are mutually exclusive.

NODATE

Suppresses printing the system date in positions one through eight of the first title line.

NOPAGE

Suppresses printing a page number.

NOHEADING

Suppresses printing column headings.

Spacing control parameters are all optional. When used, you can code them 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 Control Parameters { [SPREAD ]
      { [NOSPREAD ] +
      { [
      { [NOADJUST] +
      {
      { [NODATE] +
      { [NOPAGE] +
      { [NOHEADING] +
  
```

## REPORT Statement Example

The following program shows an example of how you can use spacing control parameters on the REPORT statement.

```

FILE PERSNL FB(150 1800)
EMP#      9 5 N
NAME     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 NAME EMP# DEPT GROSS
  
```

The program just shown produces the following report (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:

1. SEQUENCE
2. CONTROL
3. SUM
4. TITLE
5. HEADING
6. LINE

### SEQUENCE

This statement lets you 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

The format of the SEQUENCE statement is:

```
SEQUENCE field-name-1 [D] ... field-name-n [D]
```

## SEQUENCE Examples

```
SEQUENCE CO DIV DEPT GROSS-PAY D  
SEQUENCE GROUP AMT D CODE
```

**CONTROL**

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 a major to minor order.

## Syntax

The format of the CONTROL statement is:

```
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 printing totals for that field (which are still accumulated) at the corresponding control break.
- NEWPAGE, following any field or FINAL, creates a new page after printing the control break totals (or, in the case of FINAL, before printing the final totals). Page numbers continue.
- RENUM, following any field or FINAL, creates a page break and restarts page numbers at 1 after printing the control break totals (or, in the case of FINAL, before printing the final totals).

## Control Examples

```
CONTROL CO RENUM DIV DEPT NOPRINT  
CONTROL FINAL NOPRINT CO NEWPAGE DIV
```

## SUM

The SUM statement specifies the quantitative fields to total for a control report. Normally, on control reports, CA-Easytrieve/Plus totals all quantitative fields specified on the LINE statement (described 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

The SUM statement has the following format:

```
SUM quant-field-1 ... quant-field-n
```

### SUM Example

```
SUM GROSS NET
```

## TITLE

The TITLE statement lets you define a title for your report. Up to 99 titles are permitted. You can specify literals and field names on the TITLE statement.

### Syntax

The TITLE statement has the following format:

```
TITLE      [nn]  [[ ]      ] [          ]
            [[+] nn ] [field-name]
            [[-]      ] [literal  ]
            [[ ]      ] [          ]
            [COL nn ] [          ]
            [          ] [          ]
```

- You use  $\pm nn$  to alter the normal horizontal spacing between literals or fields on the title lines. The value of  $nn$  spaces are added to or subtracted from the SPACE parameter (which normally has a default of 3).
- COL  $nn$  specifies the print column number where the next title item begins. If you specify COL  $nn$ , you must also specify NOADJUST on the REPORT statement.
- If no TITLES are coded, the date and page number, which are normally automatically included in the title, do not print.

TITLE Examples

```
TITLE 01 'REPORT ONE'  
TITLE 03 'THIS PAGE FOR DIV' -2 DIV-NO  
TITLE 04 'ABC COMPANY'
```

prints

```
12/10/85          REPORT ONE          PAGE 1  
  
                THIS PAGE FOR DIV 15  
                ABC COMPANY
```

Control Field Values in Titles

Occasionally, you might 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 gives an example of this. (Output was edited for the purpose of illustration.)

Statements:

```
FILE PERSNL FB(150 1800)  
NAME      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 NAME STATE ZIP PAY-NET
```

Produce:

1/17/89	REPORT FOR THE STATE OF DC			PAGE	1
	NAME	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
	NAME	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
	NAME	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

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 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

The HEADING statement has the following format:

```
HEADING field-name ('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

The LINE statement has the following format:

```
LINE [nn]  [[ ]      ] [          ]
           [[+]     ] [field-name]
           [[-]     ] [literal  ]
           [[ ]     ] [          ]
           [COL nn ] [          ]
           [POS  nn ] [          ]
           [      ] [          ]
```

- When *literals* are specified, they print on all lines but are not used as headings.
- The value of  $\pm nn$  alters the normal spacing between line items. The value of *nn* is added to or subtracted from the SPACE parameter (which normally has a default of 3).
- *POS (position)* provides for aligning fields under the corresponding column heading positions indicated on the LINE 01 statement.
- *COL nn* specifies the print column number where the next field begins. If you specify *COL nn*, you must also specify NOADJUST on the REPORT statement.

## LINE Example

```

LINE 01 DEPT DIV NAME
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      NAME
911       02       MATT JONES
          512       2232 HILL
                   ANYWHERE IL

```

END OF FIRST READING

<b>If ...</b>	<b>Then continue with ...</b>
You completed the tutorial	Chapter 8, "Macros"
You branched to this section from the tutorial	Lesson 5 of Chapter 3, "Standard Reporting with CA-Easytrieve/Plus"

## Label Reports

YOU ARE NOW STARTING THE SECOND READING OF CHAPTER 7

You can use the CA-Easytrieve/Plus label report capability to print mailing labels and other applications that require inserting variable data in a repetitious format.

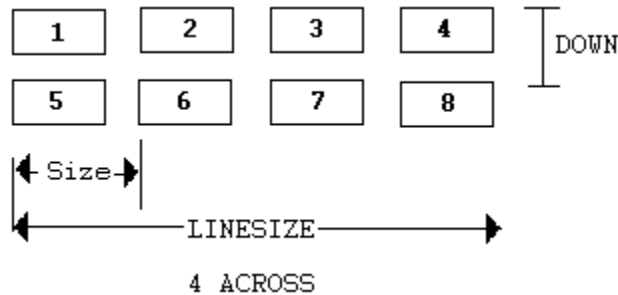
A label report is different from a standard report in the following ways:

1. Label reports do not have titles and headings.
2. You can print multiple labels side-by-side.
3. Controlled label reports permit control breaks, but do not automatically total quantitative fields. You can, however, specify totals on a SUM statement and process them in BEFORE-BREAK and AFTER-BREAK procedures (described later in this chapter).

You can use the label report function whenever each PRINT statement produces a complete logical print page.

## Label Format

Specify label reports through the LABELS option of the REPORT statement. The following example illustrates the basic label report page format:



A label line consists of one or more labels positioned across the label page. In the previous example, labels 1 through 4 compose a label line. A single line group composes each label. Therefore, CA-Easytrieve/Plus produces a label for each PRINT statement execution. CA-Easytrieve/Plus formats the labels on the page in the order shown in the above graphic.

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 that determine the type of report to print. The LABELS parameter is responsible for formatting reports that print mailing labels.

LABELS specifies that the report is 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).

## Syntax

The LABELS parameter has this format:

```
Format      {      [LABELS ]      +
Determination {      ([ACROSS nn]
Parameters   {      [DOWN nn]
              {      [SIZE nn])
```

## REPORT Statement Example

The following program creates a label report.

```

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 DOWN 6 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.

### Syntax

Testing aid parameters have this format:

```

REPORT [report-name] +
  [LIMIT integer]

```

[EVERY integer]

- The LIMIT option limits the number of records the report processes. The value of *integer* can be any integer literal in the range 1 through 32,767.
- The EVERY option lets you specify that only every *n*th line is printed in the report. The value of integer can be any integer literal in the range 1 through 32,767.

END OF SECOND READING

Please turn to Chapter 8, "Macros".

## Format Determination Parameters

YOU ARE NOW STARTING THE THIRD READING OF CHAPTER 7

Aside from the format determination parameter LABELS used (on the REPORT statement) to format label reports, there are six other format-related parameters of the REPORT statement. We describe four of them here.

Each parameter and its purpose is given in the following table:

Parameter	Purpose
DTLCTL	(Detail Control) controls the printing of control fields on detail lines of a control report.
SUMCTL	(Sum Control) controls the printing of control fields on total lines of a control report.
SUMMARY	Suppresses 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.

Syntax

The format of these four parameters is:

```

REPORT report-name +
{
  [DTLCTL { FIRST } ] +
  [ { EVERY } ]
  [ { NONE } ]
  [ ]
}
Format
Determination
Parameters {
  [ { HIAR } ] +
  [SUMCTL ( { NONE } [DTLCOPY]) ]
  [ { TAG } ]
  [ ]
}
{
  [SUMMARY] +
  [SUMFILE file-name] +
}
    
```

DTLCTL, SUMCTL, SUMMARY, and SUMFILE are described below.

## DTLCTL

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 shows an example program using DTLCTL options. You can run the program shown with any of the three options to produce the results on the following pages.

```

FILE FILE1 CARD
  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
END
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
    
```

The following example shows the use of the EVERY option of DTLCTL. Control values (contents of the STATE and ZIP fields) are printed on every detail line.

Line Description	Control Fields			Accumulator Field
	LAST-NAME	STATE	ZIP	PAY-NET
detail	BROWN	IL	60076	678.90
detail	BROWN	IL	60076	123.45
ZIP total		IL	60076	802.35
detail	JONES	IL	60077	543.21
detail	JONES	IL	60077	98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
detail	SMITH	TX	75218	666.66
detail	SMITH	TX	75218	111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

The following example shows the use of the FIRST option of DTLCTL. Control values (contents of the STATE and ZIP fields) are printed on the first detail line at top-of-page and on the first detail line after each break. This is the normal default for printing of detail lines on control reports.

Line Description	Control Fields			Accumulator Field
	LAST-NAME	STATE	ZIP	PAY-NET
detail	BROWN	IL	60076	678.90
detail	BROWN			123.45
ZIP total		IL	60076	802.35
detail	JONES	IL	60077	543.21
detail	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
detail	SMITH	TX	75218	666.66
detail	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

The following example shows the use of the NONE option of DTLCTL. Control values (contents of STATE and ZIP fields) are not printed on detail lines.

Line Description	Control Fields			Accumulator Field
	LAST-NAME	STATE	ZIP	PAY-NET
detail	BROWN			678.90
detail	BROWN			123.45
ZIP total		IL	60076	802.35
detail	JONES			543.21
detail	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
detail	SMITH			666.66
detail	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

## SUMCTL

The SUMCTL (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 with the SUMMARY parameter later in this chapter.) The following shows an example program using these options.

You can run the program shown with any of the four options to produce the results on the following pages.

```

FILE FILE1 CARD
  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
END
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
    
```

The following example shows the use of the ALL option of SUMCTL. Control values (contents of STATE and ZIP fields) are printed on all total lines.

Line Description	Control Fields			Accumulator Field
	LAST-NAME	STATE	ZIP	PAY-NET
	BROWN	IL	60076	678.90
	BROWN			123.45
ZIP total		IL	60076	802.35
	JONES	IL	60077	543.21
	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL	60077	1444.32
	SMITH	TX	75218	666.66
	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX	75218	777.77
FINAL total		TX	75218	2222.09

The following example shows the use of the HIAR option of SUMCTL. Control values (contents of STATE and ZIP fields) are printed in hierarchical order on total lines. This is the normal default for printing total lines on control reports.

<u>Line Description</u>	<u>Control Fields</u>			<u>Accumulator Field</u>
	LAST-NAME	STATE	ZIP	PAY-NET
	BROWN	IL	60076	678.90
	BROWN			123.45
ZIP total		IL	60076	802.35
	JONES	IL	60077	543.21
	JONES			98.76
ZIP total		IL	60077	641.97
STATE total		IL		1444.32
	SMITH	TX	75218	666.66
	SMITH			111.11
ZIP total		TX	75218	777.77
STATE total		TX		777.77
FINAL total				2222.09

The following example shows the use of the NONE option of SUMCTL. Control values (contents of STATE and ZIP fields) are not printed on total lines.

<u>Line Description</u>	<u>Control Fields</u>			<u>Accumulator Field</u>
	LAST-NAME	STATE	ZIP	PAY-NET
	BROWN	IL	60076	678.90
	BROWN			123.45
ZIP total				802.35
	JONES	IL	60077	543.21
	JONES			98.76
ZIP total				641.97
STATE total				1444.32
	SMITH	TX	75218	666.66
	SMITH			111.11
ZIP total				777.77
STATE total				777.77
FINAL total				2222.09

## TAG

The TAG subparameter of SUMCTL creates a line area on the left side of the total line for an annotation. This LINE 01 item is governed by the following rules:

1. The length of the area is the length of the longest control-field-name plus seven.
2. TOTAL preceded by the control field name is the annotation for control break totals.
3. FINAL TOTAL is the annotation for final totals.
4. The line item area is positioned at the left margin of the report.

The example on the following page illustrates how tags appear on a report.

Statements:

```

FILE FILE1 CARD
  LAST-NAME  1  5  A
  STATE      6  2  A
  ZIP        8  5  N
  PAY-NET   13  5  N 2
JOB INPUT FILE1 NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65 SUMCTL TAG
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE ZIP
LINE 01 LAST-NAME STATE ZIP PAY-NET
END
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
    
```

Produce:

	LAST-NAME	STATE	ZIP	PAY-NET
	BROWN	IL	60076	678.90
	BROWN			123.45
>	ZIP TOTAL			802.35
	JONES	IL	60077	543.21
	JONES			98.76
	ZIP TOTAL			641.97
>	STATE TOTAL			1444.32
	SMITH	TX	75218	666.66
	SMITH			111.11
	ZIP TOTAL			777.77
	STATE TOTAL			777.77
>	FINAL TOTAL			2222.09

## SUMMARY Reports

SUMMARY is a parameter of the REPORT statement that prints the report named on the REPORT statement as a summary report.

Summary reports consist of only total lines that normally include only control fields and totals. All detail lines are suppressed from printing.

## DTLCOPY

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 following example shows a program that produces a summary report and includes the DTLCOPY option.

**Note:** If this option were not used, the LAST-NAME values do not print.

Statements:

```
FILE FILE1 CARD
  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
END
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/Plus. You can request the summary file by defining the file in the library and then referencing it with the REPORT SUMFILE parameter.

A FILE statement must contain the file-name, record format, logical record length, and blocksize for the summary file. To use the summary file as input to a subsequent CA-Easytrieve/Plus JOB activity, you should specify the file as an unblocked VIRTUAL file. The record format can be any standard format. The record length must be large enough to contain the data that is output. Blocksize should be appropriate for the specified format and record length.

The summary file record contains three parts:

1. Control field area
2. TALLY
3. Summed field area

The *control field area* is a concatenation of the control fields specified on the CONTROL statement. The sum of the lengths of the control fields defines the length of the control field area.

TALLY is a ten-byte field that contains the value of the system defined field TALLY. TALLY accumulates the number of detail records that comprise a control break. It is a 10-byte packed decimal field with no decimal places.

The *summed fields* are concatenated to form the remaining segment of the summary file record. Each summed field is a 10-byte packed field with the same decimal specification as the source field. Therefore, the summary file record length is the sum of the control field area length, plus 10 bytes for TALLY, plus 10 times the number of summed fields.

The following example illustrates the use of SUMFILE data. The SUMMARY parameter is coded to control the printed output and does not affect the summary file. The values of SFILE are listed in order of ascending magnitude in SFILE-STATE, without reprocessing the original data:

Statements:

```

FILE FILE1 CARD
  LAST-NAME 1 5 A
  STATE     6 2 A
  ZIP       8 5 N
  PAY-NET   13 5 N 2
FILE SFILE F(30)
  SFILE-STATE 1 2 A
  SFILE-ZIP   3 5 N
  SFILE-TALLY 8 10 P 0
  SFILE-PAY-NET 18 10 P 2
*
JOB INPUT FILE1 NAME MYPROG
PRINT REPORT1
REPORT REPORT1 LINESIZE 65 +
SUMMARY SUMFILE SFILE SUMCTL DTLCOPY
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE NEWPAGE ZIP
TITLE 'REPORT FOR THE STATE OF' STATE
LINE 01 LAST-NAME STATE ZIP PAY-NET
*
JOB INPUT SFILE NAME MYPROG2
PRINT REPORT2
REPORT REPORT2 NOADJUST
SEQUENCE SFILE-STATE SFILE-PAY-NET
LINE 01 SFILE-STATE SFILE-ZIP +
        SFILE-TALLY SFILE-PAY-NET
END
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666

```

Produce:

SFILE-STATE	SFILE-ZIP	SFILE-TALLY	SFILE-PAY-NET
IL	60077	2	641.97
IL	60076	2	802.35
TX	75218	2	777.77

## Multiple Reports

### Multiple Reports to Single Printer

You can produce several reports 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)
  NAME          17  8  A
  DEPARTMENT    98  3  N
  NET           90  4  P 2
  GROSS         94  4  P 2
  DEDUCTIONS    W  4  P 2
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 NAME DEPARTMENT GROSS NET
*
> REPORT RPT2
  SEQUENCE DEPARTMENT
  TITLE 1 'REPORT TWO'
  LINE 1 DEPARTMENT NAME GROSS NET DEDUCTIONS
*
> REPORT RPT3
  CONTROL
  TITLE 1 'REPORT THREE - DEPT 911'
  LINE 1 NAME 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

You can send multiple reports in one program 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 example shows a CA-Easytrieve/Plus program that produces two reports, each sent to separate printers.

```
FILE PAYFILE
  NAME      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 LABEL-REPORT
  PRINT NORM-REPORT
END-IF
*
> REPORT LABEL-REPORT LABELS PRINTER SPFORM
SEQUENCE EMP-NUMBER
LINE 1 NAME
LINE 3 STREET
LINE 5 ADDRESS
*
> REPORT NORM-REPORT
LINE 1 NAME ADDRESS EMP-NUMBER
```

The first report declaration produces a label 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 standard report. It is output to the printer you normally use with other CA-Easytrieve/Plus programs (the default system output device).

## Report Procedures (PROCs)

Report Procedures (PROCs) are user-defined routines that are automatically invoked in a report declaration to perform special data manipulation not included in the logic subactivity. There are seven report PROCs in CA-Easytrieve/Plus.

Code 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 execute in a report PROC the same way you code logic in a JOB activity.
- No input or output is permitted.
- Although you can code these *procs* in any order, you can only code each *proc* once per report.

### REPORT-INPUT. PROC

The REPORT-INPUT. PROC permits final screening and modification of report input data. It is performed for each record selected for the report that contains the PROC.

- You must execute a SELECT statement in the PROC to cause data to continue to the report.
- If a report was SEQUENCED, this procedure is invoked after each record is output from the system sort.

## REPORT-INPUT Example

Statements:

```

FILE PERSNL
%PERSNL
  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 NAME EMP# PAY-NET PCT-NET-TO-TOT
*
> REPORT-INPUT. PROC
  PCT-NET-TO-TOT ROUNDED = PAY-NET / TOT-NET * 100
SELECT
END-PROC

```

Produce:

```

9/06/85          EXAMPLE OF REPORT-INPUT PROC          PAGE 1

```

BRANCH	EMPLOYEE NAME	EMPLOYEE NUMBER	NET 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 permits modification of totals and special annotation before total line printing caused by the CONTROL statement. You can test a system-defined field named LEVEL 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)
```

## BEFORE-BREAK Example

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:

8/20/85

THIS REPORT WILL ILLUSTRATE USE OF  
BEFORE-BREAK PROCEDURE

PAGE 1

DEPT	EMPLOYEE NUMBER	GROSS PAY	NET 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	

The BEFORE-BREAK. PROC caused the DEPARTMENT annotation at each of the breaks and modified the total in PCT to be the percent, based on total amounts.

## AFTER-BREAK. PROC

You can use an AFTER-BREAK procedure to produce special annotation on control reports. The value of LEVEL (a system-defined field) can determine which control break is processed. In the following example, the total line for the second control field ZIP receives special annotation.

### AFTER-BREAK Example

Statements:

```

FILE FILE1 CARD
  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
*
END
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 produce page footing information. It is invoked whenever end-of-page is detected. It typically produces page totals or other annotations, as in the following example of page footer annotation:

### ENDPAGE Example

```
FILE FILE1 CARD
  LAST-NAME 1 5 A
  STATE     6 2 A
  ZIP       8 5 N
  PAY-NET  13 5 N 2
JOB INPUT FILE1 NAME MYPROG
PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65 +
SUMMARY SUMCTL DTLCOPY
SEQUENCE STATE ZIP LAST-NAME
CONTROL STATE NEWPAGE ZIP
TITLE 'REPORT FOR THE STATE OF' STATE
LINE 01 LAST-NAME STATE ZIP PAY-NET
*
ENDPAGE. PROC
  DISPLAY SKIP 2 '* CONFIDENTIAL - FOR INTERNAL USE ONLY *'
END-PROC
*
END
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
```

## TERMINATION. PROC

A TERMINATION procedure is invoked at the end of the report. This procedure can print report footing information, including control totals and distribution information. The following is an example of report footing:

### TERMINATION Example

```

FILE FILE1 CARD
  LAST-NAME  1  5  A
  STATE      6  2  A
  ZIP        8  5  N
  PAY-NET   13  5  N 2
  TOTAL-NET  5  8  N 2
JOB INPUT FILE1 NAME MYPROG
  TOTAL-NET = TOTAL-NET + PAY-NET
  PRINT REPORT1
*
REPORT REPORT1 LINESIZE 65 +
  SUMMARY  SUMCTL  DTLCOPY
  SEQUENCE STATE ZIP LAST-NAME
  CONTROL  STATE NEWPAGE ZIP
  TITLE 'REPORT FOR THE STATE OF' STATE
  LINE 01 LAST-NAME STATE ZIP PAY-NET
*
TERMINATION. PROC
  DISPLAY NEWPAGE
  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
*
END
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666

```

## 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.

**Note:** When using a BEFORE-LINE/AFTER-LINE procedure, the detail line for the report is already created. You cannot modify the contents of the detail line through the BEFORE-LINE/AFTER-LINE procedures.

A BEFORE-LINE/AFTER-LINE procedure is commonly used to print an annotation before/after a detail line on the report.

## BEFORE-LINE/AFTER-LINE Example

The following example illustrates how an AFTER-LINE procedure can print information following a detail line of a report:

Statements:

```
FILE FILE1 CARD
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
*
END
```

```
BROWNIL6007612345
BROWNIL6007667890
JONESIL6007709876
JONESIL6007754321
SMITHTX7521811111
SMITHTX7521866666
```

Produce:

	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

END OF THIRD READING

Please turn to Chapter 8, "Macros."

## Introduction

Very often, programs used at the same site share certain attributes. There can be a common logic routine or, even more likely, common file and field definitions shared by many programs. To help you avoid the need for duplicated effort, CA-Easytrieve/Plus provides an easy to use, full feature macro facility.

This chapter describes the use of macros in CA-Easytrieve/Plus. In it you find:

### Reading 1

- What macros are
- Simple macro invocation

### Reading 2

- Creating macros
- Storing macros

### Reading 3

- Advanced prototyping, including use of:
  - Positional parameters
  - Keyword parameters

## Using Macros

### What is a CA-Easytrieve/Plus Macro?

A macro is really just a *chunk* of a CA-Easytrieve/Plus program and usually, this chunk has a very specific purpose.

You can store any piece of any CA-Easytrieve/Plus program you write that you want to use again in other programs in a macro library as a macro. When you want to use it in a program, you just put the name of the macro (preceded by a percent sign) in the place in your program where you need the code you stored. This means that you only have to type one line instead of retyping all of the lines that make up your macro.

### Invoking Macros

Assuming that someone at your site has already created some macros and stored them in a macro library, you can easily invoke them in your program.

To illustrate macro use, the following example shows all of the field definitions for a file called PERSNL (which we used in many of the examples in this guide). It consists of 34 lines of code.

If, however, it were stored as a *macro*, you would only need to type in one line in any of your programs that access the PERSNL file to invoke the field definitions shown.

## PERSNL File Field Definitions

The following field definitions describe fields in the PERSNL file used in many examples in this guide.

REGION	1	1	N	
BRANCH	2	2	N	
SSN	4	5	P	MASK '999-99-9999' - HEADING('SOCIAL' 'SECURITY' 'NUMBER')
EMP#	9	5	N	HEADING('EMPLOYEE' 'NUMBER')
NAME	17	20	A	HEADING 'EMPLOYEE NAME'
NAME-LAST	NAME	8	A	HEADING('LAST' 'NAME')
NAME-FIRST	NAME +8	12	A	HEADING('FIRST' 'NAME')
ADDRESS	37	39	A	
ADDR-STREET	37	20	A	HEADING 'STREET'
ADDR-CITY	57	12	A	HEADING 'CITY'
ADDR-STATE	69	2	A	HEADING 'STATE'
ADDR-ZIP	71	5	N	HEADING('ZIP' 'CODE')
PAY-NET	90	4	P 2	HEADING('NET' 'PAY')
PAY-GROSS	94	4	P 2	HEADING('GROSS' 'PAY')
DEPT	98	3	N	
DATE-OF-BIRTH	103	6	N	MASK(Y 'Z9/99/99') - HEADING('DATE' 'OF' 'BIRTH')
TELEPHONE	117	10	N	MASK '(999) 999-9999' - HEADING('TELEPHONE' 'NUMBER')
SEX	127	1	N	HEADING('SEX' 'CODE') * 1 - FEMALE * 2 - MALE
MARITAL-STAT	128	1	A	HEADING('MARITAL' 'STATUS') * M - MARRIED * S - SINGLE
JOB-CATEGORY	132	2	N	HEADING('JOB' 'CATEGORY')
SALARY-CODE	134	2	N	HEADING('SALARY' 'CODE')
DATE-OF-HIRE	136	6	N	MASK Y - HEADING('DATE' 'OF' 'HIRE')

Once stored in a macro, you can pull all of the field definitions for PERSNL into your programs with one line of code. Assuming the PERSNL field definitions were stored as a macro named PERSNL, here is an example of a program where the macro is invoked.

```

      FILE PERSNL FB (150 1800)
>    %PERSNL

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

```

The statement %PERSNL brings all of the field definitions (stored as a macro) for the PERSNL file into your program. You can see what a saving of effort this provides you if you had several programs that all accessed the PERSNL file.

## Macro Invocation Statement

The macro invocation statement consists of a macro name preceded by a percent (%) sign and followed by an optional subtype for use with DOS Source Statement Libraries. Its format is:

```
%macro-name [.subtype]
```

%macro-name

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

.subtype

For VSE users of Source Statement Libraries, subtype is an optional one-character subtype for an SSL member. It is delimited from the macro-name (book name) with a period ( . ). If subtype is not coded, the default subtype is taken from the CA-Easytrieve/Plus options table (see Appendix C of the *CA-Easytrieve/Plus Reference Guide*). The default subtype for macros stored in Source Statement Libraries is Z.

## Macro Nesting

CA-Easytrieve/Plus treats a macro invocation statement that is in the body of a macro (nested) as if it were outside of the macro. That is, no special consideration is necessary. There is no limit to the nesting level.

END OF FIRST READING

<b>If ...</b>	<b>Then continue with ...</b>
You completed the tutorial	Chapter 9, "Programming Techniques"
You branched to this section from the tutorial	Lesson 6 of Chapter 3, "Standard Reporting with CA-Easytrieve/Plus"

## Creating and Storing Macros

YOU ARE NOW STARTING THE SECOND READING OF CHAPTER 8

### Defining Macros

Macros are composed of three parts:

- The *macro prototype* statement defines the macro and its parameters.
- The *macro body* contains the CA-Easytrieve/Plus statements that are generated when the macro is invoked.
- The optional *macro termination* command ends the macro.

The name of a macro is the same as the member name (the name of the macro as it is stored) in the macro storage library. The following example shows the structure of a macro:

Prototype Statement:	<b>MACRO</b>
Body:	<pre> ***** *                                     * *  NAME:  MACRO EXAMPLE                * *                                     * *  FUNCTION:  DEFINES FIELDS IN THE PERSNL FILE.  * *                                     * ***** NAME  17  20  A  HEADING 'EMPLOYEE NAME' EMP#   9   5  N  HEADING ('EMPLOYEE' 'NUMBER') DEPT  98   3  N GROSS 94   4  P 2 HEADING ('GROSS' 'PAY') </pre>
Optional Termination Command:	<b>MEND</b>

### Prototype Statement

The prototype statement must be the first statement of a macro. It optionally defines the parameters of the macro. Its most basic format is:

```
MACRO
```

## Macro Body

The macro body consists of a series of model and actual CA-Easytrieve/Plus statements. The model statements contain one or more parameters that are replaced by the values of corresponding parameters on the prototype statement (described later in this chapter).

## Macro Termination Command

The optional macro termination command is used at the end of a macro. (When storing macros in VSE Source Statement Libraries, its use is required.) The format of the macro termination statement is:

MEND

## Storing Macros

Macro statements are stored and maintained in a macro library. The MACRO entry in the CA-Easytrieve/Plus options table (see Appendix C of the *CA-Easytrieve/Plus Reference Guide*) specifies the macro library storage access method. The types of access methods are:

---

<b>Term</b>	<b>Description</b>
PAN	Macros stored in a CA-Panvalet library and maintained through CA-Panvalet utilities.
PDS	(OS/390 only) Macros stored in a partitioned data set.
SSL	(VSE only) Macros stored in a VSE Source Statement Library. SSLs are maintained by using a variety of VSE utilities.
USER	Any user-supplied library facility that you can use for the macro library.
VSAM	A specially formatted VSAM data set that you can use as the macro library. Macros are maintained using a special CA-Easytrieve/Plus program.

---

END OF SECOND READING

Please turn to chapter 9, "Programming Techniques."

## Parameter Substitution in Macros

YOU ARE NOW STARTING THE THIRD READING OF CHAPTER 8

Sometimes you might want to store a commonly used routine in a macro. A *routine* is a series of logic statements that perform a specific task. Although the logic in your routine might never change, you can alter some of the values the routine uses.

If you had to go into a macro and make changes to values every time you wanted to use your routine, it would defeat the purpose of storing it as a macro. CA-Easytrieve/Plus provides a way of feeding values to macro routines without ever having to change the macro. This technique is called *parameter substitution*.

It lets you set up variable parameters in your macros that receive values you specify on the macro invocation statement at the time of invocation. These variable parameters consist of two types: positional and keyword parameters.

There are three basic steps to coding and using positional and keyword parameters:

1. Code positional and keyword parameters on the MACRO prototype statement.
2. Code the parameter substitution words in the body of the macro. A parameter substitution word is simply the positional parameter name preceded by an ampersand character ( & ).
3. Code the positional or keyword parameter values in your CA-Easytrieve program on the macro invocation statement.

### Positional Parameters

Positional parameters are values that are assigned in the macro according to their position on the macro prototype statement and on the macro invocation statement. They are useful when a value is always required for the parameters each time the macro is invoked. Since there is no default value for these parameters, their value must be supplied at the time of invocation. Generally, you are always required to supply some value. (In some cases, CA-Easytrieve/Plus gives the value of a null string to an unsupplied positional parameter.)

Frequently-used parameters are often coded as positional, since you need to code only the value of the parameter when invoking the macro (keyword parameters require a keyword name and a value).

## Keyword Parameters

Keyword parameters are values that are assigned in the macro according to a keyword name associated with the value. Keyword parameters are used:

- To help keep track of a large number of parameters (since they require keyword names that can be descriptive).
- For optionally used parameters (since they enable default values).

## Prototype Statement

Positional and keyword parameters appear on the prototype statement according to the following format:

```
MACRO [literal] [positional-] [keyword- default-] ...  
                [parameter-] [parameter- value]  
                [name         ] [name         ]
```

[literal]

*Literal* lets CA-Easytrieve/Plus distinguish where positional parameters end and keyword parameters begin on the statement. Replace *literal* with an integer specifying the number of positional parameters on the prototype statement. This parameter is required only when you use keyword parameters. When you specify only keyword parameters on the prototype statement, you must code a zero (0) as the value for *literal*.

[positional-parameter-name]

This is a descriptive name for the parameter. You must code positional parameters before any keyword parameters. Positional values are substituted according to their position on the prototype statement.

[keyword-parameter-name default-value]

Keyword parameters have two parts: the keyword name and the default value. The keyword name identifies the parameter; the default value is the value CA-Easytrieve/Plus uses when no value is specified in your program on the invocation statement.

## Prototype Statement Parameter Examples

The following examples show some typical prototype statements with both positional and keyword parameters.

### Positional Parameters Only

This example shows only positional parameters of the prototype statement (named POS1 and POS2).

**Note:** The number of positional parameters is not indicated.

You could have optionally coded the number of positional parameters as '2', but it is not required because no keyword parameters are present.

```
MACRO POS1 POS2
...
...
```

### Keyword Parameters Only

This example shows only keyword parameters on the prototype statement. Keyword parameters consist of the keyword name and a default value, such as KEY1 and VALUE1. The number of positional parameters is coded as zero, which is required when you use only keyword parameters.

```
MACRO 0 KEY1 VALUE1 KEY2 VALUE2
...
...
```

### Positional and Keyword Parameters

Macros with both positional and keyword parameters require that you supply positional parameters first and the number of positional parameters (in this case, 1).

```
MACRO 1 POS1 KEY1 VALUE1
...
...
```

## Macro Invocation Statement

Positional and keyword parameters appear on the macro invocation statement according to the following format:

```
%macro[-name] [positional- ] .. [keyword- keyword- ] ...
               [parameter- ] .. [parameter- parameter- ]
               [value      ] .. [name      value      ]
               [           ] .. [           ]
```

[positional-parameter-value]

This is the value that is substituted for the substitution word in the body of your macro. It is position dependent and is assigned according to the location of the positional-parameter-name on the MACRO prototype statement. This value shows up in the compile listing of your program.

[keyword-parameter-name keyword-parameter-value]

The keyword parameter name is the descriptive name that identifies the parameter and ties it to a keyword parameter name on the MACRO prototype statement.

The keyword parameter value is the value you are giving the parameter, which is then substituted for the substitution word in the body of your macro. This value shows up in the compile listing of your program.

If the keyword parameter name is specified, then a keyword parameter value must also be given and vice versa. If neither is specified, then the default value, defined on the MACRO prototype statement, is used.

## Parameter Substitution Examples

The following examples show macros that use substitution parameters. The macro prototype and invocation statements are shown with statements as they appear in the body of the macro and as they appear (in the compile listing) after substitution takes place.

### Positional Parameters

This first example demonstrates the use of positional parameters. For a macro named FILEDEF, three positional parameters are coded on the macro prototype statement:

- NAME
- TYPE
- FORMAT

These are descriptive names for file name, file type, and file format.

By entering values for these parameters on the macro invocation statement, you provide the substitution words in the macro body (&NAME, &TYPE, &FORMAT) with their content.

#### Macro Prototype Statement and Macro Body

This portion of our example represents the actual macro.

- The first line contains the prototype statement and positional parameters.
- The second line is the source code or body of the macro, with substitution words that receive their values from the invocation statement. This line is what is called into your program.

```
MACRO NAME TYPE FORMAT  
FILE &NAME &TYPE &FORMAT
```

## Macro Invocation Statement

The invocation statement shown below is what you code in your program to invoke the macro (FILEDEF) into your program. Also, it provides three values for the positional parameters shown above.

```
%FILEDEF TESTIN ' ' 'FB(150 1800)'
```

The value TESTIN is substituted for &NAME and the value FB(150 1800) is substituted for &FORMAT. The second value is simply a blank character string that is a necessary way of giving the positional parameter &TYPE no value at all.

## Resulting Code On Compile Listing

The following line of code is generated after all substitution takes place. This line of code appears on the compile listing of your program.

```
FILE TESTIN FB(150 1800)
```

Usually, positional parameters require that a value always be given on the macro invocation statement. If you do not need to use a particular positional parameter, you must code a blank character string ( ' ') as its value.

**Note:** Any single parameter that has components separated by a delimiter (such as a space) must be surrounded by single quotes.

## Keyword Parameters

In this example, keyword parameters perform the same task as in the previous example. Unlike positional parameters, keyword parameters are capable of providing a default value if no value is coded on the macro invocation statement.

You must provide default values on the macro prototype statement for each keyword name. You can use a blank character string ( ' ') as the default value, which simply substitutes *no entry* if no value is coded on the macro invocation statement.

## Macro Prototype Statement and Macro Body

This portion of our example represents the actual macro.

- The first line contains the prototype statement and keyword parameters.
- The second line is the source code or body of the macro, with substitution words that receive their values from the invocation statement. This line is what is called into your program.

```
MACRO 0 NAME FILEA TYPE ' ' FORMAT 'FB(150 1800)'  
FILE &NAME &TYPE &FORMAT
```

### Macro Invocation Statement

The invocation statement shown below invokes the macro named FILEDEF into your program. It also provides three values for the keyword parameters shown above.

```
%FILEDEF NAME TESTIN TYPE VIRTUAL FORMAT 'V(1000)'
```

### Resulting Code On Compile Listing

The following line of code is what is generated after all substitution takes place. This line of code appears on the compile listing of your program.

```
FILE TESTIN VIRTUAL V(1000)
```

You do not have to specify keywords in the same order on the macro invocation statement as on the macro prototype statement.

## Rules for Substituting Parameters

The rules for substituting macro parameters are the basic rules-of-syntax (see Chapter 4) and the following:

1. You must specify positional parameter values on the macro invocation statement in the same order as they appear on the prototype statement.
2. CA-Easytrieve/Plus gives the value of a null string to unsupplied positional parameter values. This means that the parameter is treated as nonexistent.
3. You can specify keyword parameter values in any order on the macro invocation statement.
4. CA-Easytrieve/Plus gives unsupplied keyword parameter values the default value from the prototype statement.
5. In the body of a macro, an ampersand (&) is used as a prefix on parameter substitution words to identify them. Spell parameter substitution words exactly like their counterparts on the macro prototype statement except for the leading ampersand.

Delimit parameter substitution words with a space or a period. Use the period delimiter (.) when the substituted value is concatenated with another word. CA-Easytrieve/Plus deletes the period when the parameter is replaced by its value.

**Note:** When you want to have an ampersand character in the macro body remain as an '&' character, you must code two consecutive ampersands (&&).

## Ampersands (&) and Periods (.) in Macros

The following example illustrates the use of the ampersand (&) character in the body of a macro and the use of concatenated substitution words.

**Note:** The extra '&' and the concatenation '.' characters are not part of the resulting statements.

```

                                Macro Invocation
                                Statement

...
%FILE TESTIN NEW
...

                                MACRO Prototype Statement
                                and Macro Body

MACRO NAME PREFIX
FILE &NAME
&PREFIX.-SSN 1 9 N
&PREFIX.-MAIL 10 75 A HEADING 'NAME && ADDRESS'
```

Resulting Code On Compile Listing

```

...
FILE TESTIN
NEW-SSN 1 9 N
NEW-MAIL 10 75 A HEADING 'NAME & ADDRESS'
...
```

## Instream Macros

You can also include macro statements directly in your CA-Easytrieve/Plus program. This capability is particularly useful for testing new macros before storing them in the macro library. When an instream macro has the same name as a macro in the library, the instream macro is used.

Instream macros are placed at the beginning of the source input. Each instream macro is bounded by an MSTART and MEND statement. The format of these statements is:

```
MSTART macro-name
MACRO 2 NUMBER RESULT
*****
*
* NAME:  MACRO EXAMPLE
*       CALCULATE THE CUBE OF A NUMBER
*
* FUNCTION:  THIS MACRO CALCULATES THE CUBE OF A NUMBER.
*
*****
DEFINE CUBE_NUMBER_ S 6 N VALUE 00000
CUBE_NUMBER_ = &NUMBER * &NUMBER * &NUMBER
&RESULT = CUBE_NUMBER_
MEND
```

## Operation

macro-name

Macro-name is the name of the macro. It can be from one to eight characters long. The first character must be alphabetic.

END OF THIRD READING

Please turn to chapter 9, "Programming Techniques."

# Programming Techniques

---

## Introduction

This chapter describes some CA-Easytrieve/Plus features used for debugging programs and also lists some system-defined fields you can access in your programs to do advanced programming and customization.

In this chapter, you find:

Reading 1

- Abnormal termination and diagnostics

Reading 2

- PARM statement
- Options table
- Debugging facilities

Reading 3

- System-defined fields

## Abnormal Termination

Most programming errors fall into two categories:

1. Syntax errors
2. Execution errors.

When CA-Easytrieve/Plus encounters a *syntax error*, it prints diagnostic messages that pinpoint the error, and then terminates after completing the compilation of the entire program.

When CA-Easytrieve/Plus encounters an *execution error*, it prints a diagnostic message for the error and terminates immediately.

## Diagnostic Messages

### Syntax Errors

Most of the errors made in programming are syntax errors relating to clerical mistakes or a misunderstanding of the language. The simple syntax rules and logical program structure of CA-Easytrieve/Plus nearly eliminate these errors. To pinpoint violations, CA-Easytrieve/Plus provides an extensive set of diagnostic messages (see Appendix A of the *CA-Easytrieve/Plus Reference Guide*).

#### Diagnostic Format

When CA-Easytrieve/Plus detects syntax errors in the program source code, an error message with the following format is printed:

```
(A)  (B)                (C)
-----
2*****B055 INVALID LENGTH, TYPE OR DECIMAL PLACES - 1
```

This message prints directly below the statement in error in the statement listing and has the following components:

- A. Number of the statement where the error occurred.
- B. Seven asterisks to bring attention to the error.
- C. Diagnostic message.

### Execution Errors

You can also encounter *execution errors*, most of which are easily remedied. Some of the execution errors that CA-Easytrieve/Plus intercepts include:

- Insufficient storage
- File OPEN errors
- Table file out-of-sequence.

### Statement Listing

Input to the CA-Easytrieve/Plus compiler is listed one record per line. The line consists of a CA-Easytrieve/Plus generated statement number (A), followed by the input record (B). If the input is from a macro, '-macroname' (C) is appended to the line.

The following example illustrates this:

```

(A) (B) (C)
1 FILE TESTIN
2 %DATADEF
3 REGION 1 1 N -DATADEF
4 BRANCH 1 1 N -DATADEF
5 EMP# 9 5 N. CODE 16 1 N -DATADEF
7 SEX 127 1 A -DATADEF
8 JOB INPUT(TESTIN)
9 DISPLAY NEWPAGE REGION BRANCH EMP#
10 STOP

```

END OF FIRST READING

<b>If ...</b>	<b>Then continue with ...</b>
You completed the tutorial	Chapter 4, "Library Section-Describing and Defining Data"
You branched to this section from the tutorial	Chapter 4, "Library Section-Describing and Defining Data"

YOU ARE NOW STARTING READING TWO OF CHAPTER 9

## PARM Statement and System Options Table

As mentioned in Chapter 2 of this guide, there are three sections to a CA-Easytrieve/Plus program:

- Environment
- Library
- Activity.

We deferred any further mention of the environment section until now because it is entirely optional.

The environment section of a CA-Easytrieve/Plus program consists of only one statement, PARM. The PARM statement has several parameters, all of which are optional.

Most parameters of the PARM statement have default values that are set in the System Options Table at the time CA-Easytrieve/Plus is installed on your system. (System options are listed in Appendix C of the *CA-Easytrieve/Plus Reference Guide*.) When defaults are satisfactory for achieving the results you want, you do not need to code the PARM statement. However, if you prefer to override pre-determined system options, you can do it with the PARM statement.

## Syntax

PARAM statement parameters useful in the program debugging process have the following format:

```
PARAM +
[ { SNAP } ]
[ ABEXIT { NOSNAP } ] +
[ { NO } ]

[ DEBUG ( [ CLIST ] [ PMAP ] [ DMAP ] [ FLDCHK ] [ FLOW ] [ FLOWSIZ literal-1 ] +
[ [ NOCLIST ] [ NOPMAP ] [ NODMAP ] [ NOFLDCHK ] [ NOFLOW ]
[ [ [ STATE ] [ XREF { LONG } ] ] ]
[ [ NOSTATE ] [ NOXREF { SHORT } ] ] ) ] +
[ [ [ ] [ ] ] ]

[ LIST ( [ FILE ] [ PARM ] ) ]
[ [ NOFILE ] [ NOPARM ] ]
[ [ ] [ ] ]
```

## PARAM Statement Parameters

### ABEXIT

ABEXIT indicates the level of control exercised over program interrupt codes 1 through 11 (see the appropriate *Principles of Operation* guide for information on these codes).

- SNAP prints a formatted dump of CA-Easytrieve/Plus storage areas with an error analysis report.
- NOSNAP prints only an error analysis report.
- NO inhibits CA-Easytrieve/Plus interception of program interrupts.

### DEBUG

DEBUG and its subparameters control generation of certain system outputs. These outputs analyze programming errors that cause abnormal execution termination. Subparameters prefixed with NO inhibit the named operation.

#### CLIST

CLIST creates a condensed listing of the executable program produced by the compiler.

#### PMAP

PMAP creates a complete listing of the executable program produced by the compiler.

CLIST and PMAP are mutually exclusive subparameters.

## DMAP

DMAP creates a listing of the data map for each file and its associated fields.

## FLDCHK

FLDCHK validates all data references during program execution. A data reference is invalid if a field-name was referenced in a file that had no active record. Invalid references, that is, data reference after end-of-file, might otherwise cause a program interruption or incorrect program results.

## FLOW

FLOW activates a trace of the statements being executed. The statement numbers print in the associated analysis report during an abnormal termination.

FLAWSIZ literal-1

FLAWSIZ establishes the number of entries in the trace table for the flow option. Literal-1 is a numeric value from 1 to 4096.

## STATE

STATE saves the statement number of the statement currently executing. The statement number is then printed in the associated abnormal termination messages.

## XREF

XREF creates a cross reference listing of each field name, file name, procedure name, segment name, report name, and statement label.

- LONG implies that entries are listed, even though they are not referenced.
- SHORT lists only referenced entries.

## LIST

LIST controls the printing of certain system outputs. Subparameters prefixed with NO inhibit the named operation.

## FILE

FILE prints file statistics at the end of each JOB or SORT activity.

## PARM

PARM prints system parameters at the conclusion of the syntax check operation.

## PARM Examples

The following examples illustrate typical uses of the PARM statement:

PARM for production ...

```
PARM LINK(MYPROG) DEBUG(CLIST, PMAP, DMAP) +
      SORT (MSG (ALL, PRINTER))
FILE PERSNL FB(150 1800)
%PERSNL
JOB INPUT PERSNL NAME MYPROG
      PRINT REPORT1
*
REPORT REPORT1
LINE NAME DEPT
```

PARM for program testing ...

```
PARM ABEXIT (SNAP) +
      DEBUG (PMAP, DMAP FLDCHK, FLOW, +
            FLOWSIZ (20), STATE) VFM (10)
FILE PERSNL FB(150 1800)
%PERSNL
JOB INPUT PERSNL NAME MYPROG
      PRINT REPORT1
*
REPORT REPORT1
LINE NAME DEPT SALARY-CODE
```

END OF SECOND READING

Please turn to Chapter 4, "Library Section-Describing and Defining Data."

## System-Defined Fields

YOU ARE NOW STARTING READING 3 OF CHAPTER 9

System-defined fields are fields that CA-Easytrieve/Plus automatically defines and maintains internally. You can access these fields in your program to retrieve data that can be useful in processing or certain types of error trapping.

CA-Easytrieve/Plus provides three categories of system-defined fields:

- General purpose fields
- File processing fields
- Report processing fields.

## General Purpose Fields

### SYSDATE

SYSDATE is a read only eight-byte alphanumeric field that contains the system date at the start of CA-Easytrieve/Plus execution. The DATE option of the Options Table (see your *CA-Easytrieve/Plus Reference Guide*) determines the format of the date. A slash (/) separates the month, day, and year components of the date; for example, MM/DD/YY.

### SYSTIME

SYSTIME is a read only eight-byte alphanumeric field that contains the system time at the start of CA-Easytrieve/Plus execution. A slash (/) separates the data into hours, minutes, and seconds; for example, HH/MM/SS.

### RETURN-CODE

RETURN-CODE is a four-byte binary field whose contents are returned to the operating system in register 15 when CA-Easytrieve/Plus terminates. RETURN-CODE is initialized to zero, but you can set it to any value. RETURN-CODE applies only to OS/390 systems.

## File Processing Fields

When you use these fields, you find that each file has its own unique processing fields.

### RECORD-LENGTH

RECORD-LENGTH is a two-byte binary 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.

### RECORD-COUNT

RECORD-COUNT is a read-only four-byte binary field with zero decimal places that contains the number of logical I/O operations performed on a file.

### FILE-STATUS

FILE-STATUS is a read-only field that contains the results of the most recent I/O operation on a file.

## Report Processing Fields

### 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 in report procedures. TALLY is commonly used to determine averages for a control level.

TALLY is a ten-byte packed decimal field with zero decimal places. This definition is used for calculations contained in report procedures. The TALLYSIZE parameter of the REPORT statement defines the number of digits that 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 field is defined as a two-byte binary field.
- 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

# Index

## A

---

Abnormal termination, 9-1

Access to levels, 7-32

Activities  
    JOB, 2-9  
    SORT, 2-10

Activity definition section, 2-9, 3-7

Addition, 5-7

AFTER-BREAK procedure, 7-34

AFTER-LINE procedure, 7-36

Alphabetic literals, 4-5

Ampersand in macros, 8-13

Arithmetic operations, 5-7

Arithmetic operators, 4-6, 5-3

Assignment statement, 5-8

Asterisk  
    arithmetic operator, 4-6  
    to designate comments, 4-2

Automatic input and output, 6-2

Automatic input using the JOB statement, 5-2, 6-2

Automatic totals on reports, 3-21, 7-10

## B

---

BEFORE procedure with SORT, 5-15

BEFORE-BREAK procedure, 7-32

BEFORE-LINE procedure, 7-36

Blank when zero (BWZ), 4-11

Blank When Zero (BWZ), 3-16

Blocksize, 3-3

Branching, 5-12

Breaks, control, 3-21, 7-10

BWZ, 3-16, 4-11

## C

---

CA-Easytrieve Plus  
    benefits, 2-4  
    keywords, 4-4  
    program structure, 2-9  
    syntax rules, 4-2  
    words, 4-3

Calculations, 5-7

Column headings on reports, 3-16, 7-13

Comments, 4-2

Comparing against a range of values, 5-3

Comparing field values, 5-3

Conditional expressions, 3-9, 5-3  
    arithmetic operators, 5-3  
    combining, 5-6  
    comparing two fields, 5-3  
    IF/ELSE, 5-4

Continuation characters, 4-3

Control breaks, 3-21, 7-10, 7-12  
    access to levels, 2-6  
    overriding totals with SUM, 7-11

Control field values in titles, 7-12

CONTROL statement, 3-21, 7-10

---

COPY statement, 4-21  
Copying edit masks, 3-16, 4-11  
Creating and storing macros, 8-5  
Customizing column headings, 3-16, 7-13

## D

---

Data definition, 3-3, 4-6, 4-7, 4-8, 4-9, 4-10, 4-11, 4-12, 4-13  
Data types, 3-5, 4-9  
Decimal positions, 3-5, 3-6, 4-9  
Declarative statements, reporting, 3-19, 7-9  
DEFINE statement, 3-4, 4-8, 4-9  
    HEADING parameter, 3-16, 4-10  
    in an activity, 4-13  
    MASK parameter, 3-14, 3-15, 3-16, 4-10  
DEFINE Statement  
    RESET parameter, 4-17  
Defining fields, 3-4, 4-6, 4-8, 4-9, 4-10, 4-12, 4-13  
    working storage, 3-10, 4-13, 4-15  
Defining files, 3-3, 4-6  
Defining report declaration, 3-13  
Delimiters, 4-3  
Describing files and fields, 4-6  
Description (DESC), tables, 5-19  
Detail lines on reports, 3-11, 3-24  
Diagnosing an error, 3-27  
Diagnostic errors  
    statement listing, 9-2  
Diagnostic messages  
    execution errors, 9-2  
    syntax errors, 9-2  
DISPLAY statement, 6-5  
Division, 5-7  
DO loops, 5-11  
DO WHILE statement, 5-11  
    nesting example, 5-12  
Documentation conventions, 1-3

Dollar signs, 3-15  
DTLCOPY parameter, 7-25  
DTLCTL (detail control) parameter, 7-20  
duplicating field definitions, 4-21

## E

---

Edit masks, 3-14, 4-10  
    decimals and commas, 3-15, 4-10  
    digits, 3-15, 4-10  
    dollar signs, 3-15, 4-10  
    high-order zeros, 3-15, 4-10  
    naming, 3-15, 4-11  
    rules of use, 3-15, 4-10  
Editing report output, 4-10  
Eeport procedures  
    AFTER-LINE, 7-36  
END-DO statement, 5-11  
ENDPAGE procedure, 7-35  
END-PROC keyword, 5-17  
Environment definition section, 2-9, 9-3  
Error messages, 3-27, 9-2  
EXECUTE parameter, 5-13  
Execution errors, 9-1, 9-2  
EXIT parameter of FILE statement, 4-20  
Exit routines, 4-20  
Explicit redefinition of fields, 4-17  
expressions  
    conditional, 5-3  
Expressions  
    arithmetic, 5-7  
    conditional, 3-9  

## F

---

Field headings on reports, 3-16, 7-4  
fields  
    duplicating definitions using COPY, 4-21

---

## Fields

- characteristics, 3-4
- characteristics of, 4-8
- data type, 3-5, 3-6, 4-9
- decimal positions, 4-9
- definition, 4-8, 4-9, 4-10, 4-12, 4-13
- definition, 3-4
- definition of, 4-6
- edit masks, 4-10
- headings on reports, 4-10
- length, 3-5, 4-9
- naming, 4-8
- naming, 3-5
- naming of, 4-4
- qualifying, 4-4
- redefinition, 4-17
- start location, 4-8
- static working storage, 4-15
- working storage, 4-13

## File directing parameters of REPORT, 7-29

### FILE statement, 3-3, 4-7

- EXIT parameter, 4-20
- MODIFY parameter, 4-20
- NR parameter, 4-20
- user exit routines, 4-20
- VIRTUAL parameter, 4-19

## Files

- definition, 3-3
- definition of, 4-6, 4-7
- layout example, 4-6

## Fixed blocked records, 3-3, 4-20

## Floating dollar signs, 3-15, 4-10

## Flowchart symbols, 2-3

## Format determination parameters, 7-19

## Format determination parameters of REPORT, 7-16

- DTLCTL, 7-20
- SUMCTL, 7-21
- TAG, 7-24

## FROM parameter, 6-8

---

## G

## GET statement, 6-7

## GOTO statement, 5-12

## Grand totals on reports, 3-21, 7-10

## Grouping like data on reports, 3-21, 7-10

---

## H

## HEADING parameter, 3-16, 3-17, 4-10

## HEADING statement, 3-23, 7-13

## Headings on reports, 3-23, 4-10, 7-4

## Hexadecimal literals, 4-5

## Hyphen

- arithmetic operator, 4-6
- continuation character, 4-3

---

## I

## Identifiers, 4-6

## IF statement, 3-9, 5-3

- IF/ELSE, 5-4
- special uses, 5-5

## Initializing working storage, 4-17

## INPUT parameter, 5-2

## Input to a program, 6-2

- automatic, 5-2, 6-2
- controlled, 6-7
- GET statement, 6-7
- POINT statement, 6-9
- READ statement, 6-10

## Instream macros, 8-13

## Instream tables, 5-20

## INTEGER parameter, 5-8

---

## J

## JOB Activities, 5-2

## JOB activity, 2-10, 5-2

- naming, 5-2

## Job Control Language (JCL) statements, 7-29

## JOB statement, 3-7, 5-2

- INPUT parameter, 5-2
- NAME parameter, 5-2

---

## K

---

Keyword parameters, 8-8

Keyword parameters in macros, 8-7

Keywords, 4-4

## L

---

Label Reports, 7-15

Labels, 4-5

format, 7-15

Length of fields, 4-9

Length of print line on reports, 3-19, 7-5, 7-7

Lesson 1

Your First CA-Easytrieve Plus Report Program,  
3-2

Lesson 2

Expanding Your First Report Program, 3-7

Lesson 3

Printing CA-Easytrieve Plus reports, 3-13

Lesson 4

Report Declarations, 3-18

Lesson 5: Making Your Job Easier with Macros, 3-25

Lesson 6: Diagnosing An Error, 3-27

Library definition section, 2-9, 4-6

Line groups on reports, 7-5

Line item positioning on reports, 7-5

LINE statement, 3-11, 3-24, 7-14

LINESIZE parameter of REPORT, 3-19, 7-7

Literals

alphabetic, 4-5

hexadecimal, 4-5

numeric, 4-5

Logic, 3-8

logic in programs, 5-3

Logic in programs

assignments, 5-8

branching, 5-12

calculations, 5-7

conditional expressions, 5-3

loops, 5-11

moves, 5-10

Logical connectors, 5-6

Loops, 5-11

## M

---

Macro invocation statement, 8-4

MACRO prototype statement, 8-5

parameter substitution, 8-8

Macros

creating and storing, 8-5

definition, 8-2

instream, 8-13

invocation, 8-2

nesting of, 8-4

parameter substitution, 8-7

prototype statement, 8-5

termination command, 8-5

use, 3-25

Making your job easier with macros, 3-25

MASK parameter, 3-14, 4-10, 4-11

BWZ, 3-16, 4-11

MEND command, 8-6

minus sign, 4-6

MODIFY parameter, 4-20

MOVE statement, 5-10

Moving data, 5-10

Multiple parameters, 4-4

Multiple reports

to a single printer, 7-28

to more than one printer, 7-29

Multiple statements on one line, 4-2

Multiplication, 5-7

## N

---

NAME, 3-8

Naming a job activity, 5-2

---

Naming a JOB activity, 3-8  
Naming edit masks, 3-16, 4-11  
Naming of fields, 4-4  
Naming reports, 3-19  
Numeric literals, 4-5

## O

---

Order of report definition statements, 3-19  
Output  
    DISPLAY statement, 6-5  
    PUT statement, 6-8  
    WRITE statement, 6-11  
Output files  
    implicit start locations, 4-19  
Overlay redefinition of fields, 4-18  
Overriding control break totals, 3-22, 7-11

## P

---

Parameter substitution in macros, 8-7  
Parameters, multiple, 4-4  
Parentheses  
    in calculations, 5-7  
    to indicate group relationships, 4-4  
PARM statement and system option table, 9-3  
Percent symbol  
    macro invocation, 3-25  
Percent symbol, macro invocation, 8-4  
Period  
    concatenation in macros, 8-13  
    statement delimiter, 4-2  
Plus sign  
    arithmetic operator, 4-6  
    continuation character, 4-3  
POINT statement, 6-9  
Positional parameters, 8-7  
Positional parameters in macros, 8-7

Pre-printed forms, 2-6, 7-15  
Print line, length, 3-19  
Print line, length of, 7-7  
PRINT statement, 3-13, 6-2  
    execution flow, 6-3  
    work file processing, 6-4  
Printing  
    detail lines on report, 3-11  
Printing CA-Easytrieve Plus Reports, 6-2  
PROC keyword, 5-17  
Procedures  
    nesting, 5-17  
    user defined, 5-16  
Processing activities, 3-7, 5-2  
Program examples, format of, 2-3  
Program input, 3-7, 6-2  
Program logic, 3-8, 5-3  
    assignments, 5-8  
    branching, 5-12  
    calculations, 5-7  
    conditional expressions, 5-3  
    loops, 5-11  
    moves, 5-10  
Program termination, 5-13  
Programming errors  
    execution errors, 9-1  
    syntax errors, 9-1  
Publications, CA-Easytrieve Plus, 1-2  
PUT statement, 6-8

## Q

---

Qualifying fields, 4-4

## R

---

Random access processing, 6-10  
READ statement, 6-10

---

Records  
  definition, 3-4  
  fixed, blocked, 3-3, 4-20  
  length and type, 4-20

Redefining fields  
  explicit redefinition, 4-17  
  overlay redefinition, 4-18

Report declaration, 3-13

Report declarations, 3-18

Report definition statements, 3-19, 7-9  
  CONTROL, 3-21, 7-9  
  HEADING, 3-23, 7-9  
  LINE, 3-24, 7-9  
  order, 3-19  
  SEQUENCE, 3-20, 7-9  
  SUM, 3-22, 7-9  
  TITLE, 3-22, 7-9

Report procedures  
  AFTER-BREAK, 7-34  
  BEFORE-BREAK, 7-32  
  BEFORE-LINE, 7-36  
  ENDPAGE, 7-35  
  REPORT-INPUT, 7-30  
  TERMINATION, 7-36

Report procedures (PROCs), 7-30

Report processing, 7-6

REPORT statement, 3-19, 7-6, 7-7  
  format determination parameters, 7-16  
  LINESIZE, 3-19, 7-7  
  report name, 3-19  
  testing aid parameters, 7-18

Report subactivities, 2-10

REPORT-INPUT. PROC, 7-30

reports, 7-2

Reports  
  annotating, 7-36  
  control breaks, 2-6, 3-21, 7-10  
  creation, 7-2  
  creation, 3-19  
  detail lines, 3-24, 7-2, 7-5  
  editing, 3-14, 4-10  
  headings, 3-23, 7-2  
  line groups, 7-2, 7-5  
  output, 2-6, 6-2  
  printing, 3-13  
  printing of, 6-2  
  procedures, 2-6  
  sorting, 3-20, 7-9  
  titles, 3-22, 7-2, 7-11  
  totals, 3-21, 7-10, 7-11  
  totals on, 7-10

RESET parameter of DEFINE statement, 4-17

RETAIN parameter of FILE statement, 4-20

ROUNDED parameter, 5-8

Rounding fractional values, 5-8

Routines  
  exit, 4-20

Rules of syntax, 4-2

---

## S

---

Search argument (ARG), 5-19

Sections of a program  
  activity, 2-9, 3-7  
  environment, 2-9  
  library, 2-9, 3-3

SEQUENCE statement, 3-20, 7-9

Sequential file processing, 6-5

Size of print line on reports, 3-19, 7-5

Slash, division symbol, 4-6

SORT activity, 2-10

SORT statement, 5-14  
  procedures, 5-15

Sorting program input, 5-14

Sorting report data, 3-20, 7-9

Spacing control parameters of REPORT, 7-7

Special IF statements, 5-5

Stack a column, 4-10

Standard Reports, 7-2

START/FINISH procedures, 5-18

Starting position of fields, 4-8

Statement listing, 9-2

---

Statements  
  area, 4-2  
  comment, 4-2  
  continued, 4-3  
  delimiters, 4-3  
  labels, 4-5  
  multiple, 4-2

Static working storage fields, 4-15

STATUS parameter, 6-9

STOP statement, 5-13

Structure of a CA-Easytrieve Plus program, 2-9

Subtraction, 5-7

SUM statement, 3-22, 7-11

SUMCTL (sum control) parameter, 7-21

Summary files, 7-26

SUMMARY reports, 7-25

Summing Things Up  
  tutorial summary, 3-28

Suppressing zeros, 3-15, 3-16, 4-11

Syntax errors, 9-1, 9-2

Syntax rules, 4-2

System option table, 9-3

System-defined fields  
  FILE-STATUS, 9-7  
  LEVEL, 9-8  
  RECORD-COUNT, 9-7  
  RECORD-LENGTH, 9-7  
  RETURN-CODE, 9-6  
  SYSDATE, 9-6  
  SYSTEME, 9-6  
  TALLY, 9-8

## T

---

Tables, 5-19  
  ARG, 5-19  
  DESC, 5-19  
  external, 5-21  
  instream, 5-20  
  SEARCH statement, 5-21  
  testing for a match, 5-21

TAG subparameter of SUMCTL, 7-24

Temporary work files, 2-7, 4-20

Terminating program execution, 5-13

TERMINATION procedure, 7-36

Testing aid parameters, 7-18

TITLE statement, 3-22, 7-11  
  control field values in titles, 7-12

Titles on reports, 3-22, 7-3

Totals on reports, 3-21, 7-10  
  overriding with SUM, 3-22, 7-11

Tutorial, 3-1  
  lessons, 3-1

## U

---

User Controlled Input and Output, 6-5

User defined procedures, 5-16

User exit routines, 4-20

User Guide  
  structure, 2-1

Using macros, 8-2

## V

---

Virtual file manager (VFM), 2-7

Virtual File Manager (VFM), 4-19

VIRTUAL parameter of FILE statement, 4-19

## W

---

Words in a statement, 4-3

Work files  
  establishing temporary work files with VFM,  
  4-20

Working storage, 3-10  
  initialization, 4-17  
  type S (static), 4-15  
  type W, 3-10, 4-13

WRITE statement, 6-11

---

## **X**

---

Xontrol breaks  
overriding totals with SUM, 3-22

## **Z**

---

Zero suppress, 3-15, 3-16, 4-11