

# **Advantage™ CA-Easytrieve® Plus Report Generator Toolkit**

## **Macro Reference Guide**

**2.0**



Computer Associates®

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

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

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

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

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

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

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

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

© 2001 Computer Associates International, Inc.

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

# Contents

## Chapter 1: Introduction

Capabilities .....	1-2
Related Publications .....	1-2
CA-Easytrieve Plus .....	1-2
Command Notation .....	1-3

## Chapter 2: Using Routines

Generalized Routines .....	2-1
Integrity Tests .....	2-1
Date and Time Routines .....	2-2
Numeric Routines .....	2-3
File Comparison .....	2-3
Test Data Generation .....	2-7
Miscellaneous Routines .....	2-8

## Chapter 3: Coding Guidelines

Types of Toolkit Routines .....	3-1
Inline Routines .....	3-2
Stand-alone Routines .....	3-4
Stand-alone DISPLAY and Stand-alone REPORT Routines .....	3-4
Examples .....	3-8

## Chapter 4: Generalized/Statistical Routines A-D

ADDRCMP .....	4-1
Syntax .....	4-1
Operation - Stand-alone REPORT .....	4-3
Operation - Database .....	4-3
Examples .....	4-3

---

ALPHACON .....	4-5
Syntax .....	4-6
Operation - Inline .....	4-6
Operation - Database .....	4-6
Examples .....	4-7
ALPHAGEN .....	4-8
Syntax .....	4-8
Operation - Inline .....	4-9
Operation - Database .....	4-9
Examples .....	4-9
APR .....	4-10
Syntax .....	4-11
Operation - Inline .....	4-11
Operation - Database .....	4-11
Example .....	4-12
BADGEN .....	4-13
Syntax .....	4-13
Operation - Inline .....	4-13
Operation - Database .....	4-13
Examples .....	4-14
CONVAE .....	4-16
Syntax .....	4-16
Operation - Inline .....	4-17
Operation - Database .....	4-17
Example .....	4-18
CONVEA .....	4-18
Syntax .....	4-18
Operation - Inline .....	4-19
Operation - Database .....	4-20
Example .....	4-20
DATECALC .....	4-20
Syntax .....	4-20
Operation - Inline .....	4-22
Operation - Database .....	4-22
Example .....	4-23
DATECONV .....	4-23
Syntax .....	4-23
Operation - Inline .....	4-25
Operation - Database .....	4-25
Example .....	4-25

---

DATEGEN .....	4-26
Syntax .....	4-26
Operation - Inline .....	4-28
Operation - Database .....	4-28
Examples .....	4-28
DATEVAL .....	4-29
Syntax .....	4-29
Operation - Inline .....	4-30
Operation - Database .....	4-31
Example .....	4-31
DAYSAGO .....	4-32
Syntax .....	4-32
Operation - Inline .....	4-33
Operation - Database .....	4-34
Example .....	4-34
DAYSAGOL .....	4-35
Syntax .....	4-35
Operation - Inline .....	4-36
Operation - Database .....	4-37
Example .....	4-37
DAYSALC .....	4-38
Syntax .....	4-38
Operation - Inline .....	4-40
Operation - Database .....	4-40
Example .....	4-40
DIVIDE .....	4-41
Syntax .....	4-41
Operation - Inline .....	4-41
Operation - Database .....	4-41
Example .....	4-42
DOLUNIT .....	4-42
Syntax .....	4-43
Operation - Stand-alone DISPLAY .....	4-46
Operation - Database .....	4-47
Example .....	4-47
DUPTTEST .....	4-49
Syntax .....	4-49
Operation - Stand-alone REPORT .....	4-50
Operation - Database .....	4-51
Example .....	4-51

---

## Chapter 5: Generalized/Statistical Routines E-W

EACHNTH .....	5-1
Syntax .....	5-1
Operation - Stand-alone DISPLAY .....	5-2
Operation - Database .....	5-3
Example .....	5-3
EXPO .....	5-3
Syntax .....	5-3
Operation - Inline .....	5-4
Operation - Database .....	5-5
Example .....	5-5
FILECOMP .....	5-6
Syntax .....	5-6
Operation - Stand-alone DISPLAY .....	5-8
Operation - Database .....	5-8
Examples .....	5-8
FILEGEN .....	5-11
Syntax .....	5-11
Operation - Inline .....	5-11
Operation - Database .....	5-12
Example .....	5-12
FLDVALR .....	5-12
Syntax .....	5-12
Operation - Stand-alone REPORT .....	5-14
Operation - Database .....	5-14
Example .....	5-15
FLDVALT .....	5-15
Syntax .....	5-15
Operation - Stand-alone REPORT .....	5-17
Operation - Database .....	5-18
Example .....	5-18
FLDVALV .....	5-19
Syntax .....	5-19
Operation - Stand-alone REPORT .....	5-20
Operation - Database .....	5-21
Example .....	5-21
GAPCHCK .....	5-22
Syntax .....	5-22
Operation - Stand-alone REPORT .....	5-23
Operation - Database .....	5-23
Example .....	5-23

---

GETDATE.....	5-24
Syntax .....	5-24
Operation - Inline .....	5-24
Operation - Database .....	5-24
Example .....	5-24
GETDATEL .....	5-25
Syntax .....	5-25
Operation - Inline .....	5-25
Operation - Database .....	5-25
Example .....	5-25
MULTDUP .....	5-26
Syntax .....	5-26
Operation - Stand-alone REPORT .....	5-27
Operation - Database .....	5-28
Examples .....	5-28
NUMGEN.....	5-30
Syntax .....	5-30
Operation - Inline .....	5-31
Operation - Database .....	5-31
Examples .....	5-31
NUMTEST .....	5-32
Syntax .....	5-32
Operation - Inline .....	5-33
Operation - Database .....	5-33
Example .....	5-33
RANDOM.....	5-34
Syntax .....	5-34
Operation - Inline .....	5-34
Operation - Database .....	5-34
Example .....	5-35
RANDSPAN .....	5-35
Syntax .....	5-35
Operation - Inline .....	5-36
Operation - Database .....	5-36
Example .....	5-36
SQRT .....	5-37
Syntax .....	5-37
Operation - Inline .....	5-37
Operation - Database .....	5-37
Example .....	5-38

---

SRCECOMP .....	5-38
Syntax .....	5-38
Operation - Stand-alone REPORT .....	5-39
Operation - Database .....	5-40
Example .....	5-40
STDDEV .....	5-41
Syntax .....	5-41
Operation - Inline .....	5-43
Operation - Database .....	5-43
Examples .....	5-44
TIMECONV .....	5-47
Syntax .....	5-47
Operation - Inline .....	5-48
Operation - Database .....	5-48
Example .....	5-48
UNBYTE .....	5-49
Syntax .....	5-49
Operation - Inline .....	5-50
Operation - Database .....	5-50
Example .....	5-50
WEEKDAY .....	5-51
Syntax .....	5-51
Operation - Inline .....	5-52
Operation - Database .....	5-53
Example .....	5-53

## Chapter 6: Basic SMF Reporting Facility

SMF Audit Routines .....	6-1
IPL Reporting - SMF000 .....	6-4
Abnormal Step Termination - SMF004 .....	6-6
Abnormal Job Terminations - SMF005 .....	6-11
Control Output Forms - SMF006 .....	6-16
SMF Data Lost - SMF007 .....	6-18
Data Set Activity - SMF014 .....	6-19
Scratched Data Sets - SMF017 .....	6-23
Renamed Data Sets - SMF018 .....	6-26
Job Initiations - SMF020 .....	6-28
JES2 and JES3 Integrity - SMF049 .....	6-30
VSAM Opens - SMF062 .....	6-31
VSAM Deletes - SMF067 .....	6-33

---

VSAM Renames - SMF068 .....	6-36
Frequency Distribution of SMF Records - SMFCNT .....	6-37
SMF Record Field Definitions .....	6-39
Operation .....	6-40

## Chapter 7: Advanced SMF Reporting Facility (JIF)

JIF Capabilities .....	7-1
Facility Description .....	7-2
JIFOPTS .....	7-2
JIFSEL .....	7-2
User Exit Facility .....	7-2
Read Input Exit .....	7-2
Statistical Reporting Routines .....	7-2
Facility Operation .....	7-3
MVS/XA 2.2.0 Users .....	7-3
MVS/ESA 3.1.3 Users .....	7-4
JIFOPTS .....	7-4
JIFSEL .....	7-5
SMF Record Selection .....	7-5
JIFSEL Data Sets .....	7-9
User Exit Facility .....	7-23
EXIT1 .....	7-24
EXIT2 .....	7-24
Read Input Exit .....	7-24
Using the Exit .....	7-25
Using the Availability Flags .....	7-25
XFLAG Definitions .....	7-26
Example .....	7-26
Statistical Reporting Routines .....	7-27
Syntax .....	7-28
Building Customized Routines .....	7-28
Statistical Routines .....	7-29
Service Unit Distribution - OSJIF01 .....	7-29
Performance Objective Summary - OSJIF02 .....	7-31
Workload Trend Analysis - OSJIF03 .....	7-33
Performance Group Profile - OSJIF04 .....	7-34
Performance Group Summary - OSJIF05 .....	7-35
Performance Group/Priority/Class - OSJIF06 .....	7-37
Hourly Throughput Analysis - OSJIF07 .....	7-38
Service Requirements - OSJIF08 .....	7-39

---

Hourly Turnaround - OSJIF09 .....	7-41
Page Peaking Periods .....	7-42
Class Structure Analysis - OSJIF11 .....	7-44
CPU Time Distribution - OSJIF12 .....	7-45
Tape Allocation - OSJIF13 .....	7-46
Application Trend Analysis - OSJIF14 .....	7-48
Abends by Abend Code - OSJIF15 .....	7-50
Abend by Program - OSJIF16 .....	7-52
Abend by Programmer - OSJIF17 .....	7-54
Audit File Statistical Report - OSJIF18 .....	7-56
TSO Session Analysis - OSJIF19 .....	7-57
Weighting and Costing Examples .....	7-59
Building Customized Routines .....	7-60
Job Priority Weighting - EXWGTPTY .....	7-60
CPU Weighting - EXWGTCPU .....	7-61
CPU Charging - EXCHGCPU .....	7-62
I/O Charging - EXCHGIO .....	7-63
TPUT and TGET Charging - EXCHGTPG .....	7-65
Unit Record Device Charging - EXCHGUR .....	7-66
Combined Costs - EXCOSTS .....	7-67
Billing Routines Examples .....	7-68
Invoice Ledger - JIFBEX01 .....	7-69
Detail Charge Audit - JIFBEX02 .....	7-70
Job Charge Summary - JIFBEX03 .....	7-72
Data Center Cost Recovery - JIFBEX04 .....	7-73
Monthly Utilization by Cost Center - JIFBEX05 .....	7-75
Data Processing Invoice - JIFBEX06 .....	7-76
TSO User Charging Report - JIFBEX07 .....	7-77

## Chapter 8: Graphing Facility

Using the Facility .....	8-1
Coding .....	8-2
Restrictions .....	8-2
Required Statements .....	8-3
FILE Statements .....	8-3
Graphing Facility Invocation Statement .....	8-3
Syntax .....	8-3
Operations .....	8-4
Example .....	8-5

---

Bar Graph .....	8-6
Syntax .....	8-6
Operation .....	8-6
Keywords .....	8-7
Examples .....	8-10
Histogram .....	8-14
Syntax .....	8-14
Operation .....	8-14
Keywords .....	8-14
Example .....	8-17
Plot .....	8-19
Syntax .....	8-19
Operation .....	8-19
Keywords .....	8-20
Example .....	8-22
Deviation Bar Graph .....	8-24
Syntax .....	8-24
Operation .....	8-24
Keywords .....	8-24
Example .....	8-27
Example OS/390 and z/OS JCL .....	8-28

## Chapter 9: Advanced Techniques

CA-Easytrieve Plus Procedures .....	9-2
Example - Procedure with an Inline Routine .....	9-3
Stand-alone Routines (DISPLAY and REPORT) .....	9-4
Logic-After-Invocation of Stand-alone Routines .....	9-4
Logic-Before-Invocation of Stand-alone Routines .....	9-9
Use of Sample Flags .....	9-12
Stacking Stand-alone Routines .....	9-17
Limitations .....	9-17
First Routine Limitation .....	9-18
Changing Parameter Limitation .....	9-18
Database Use of Toolkit .....	9-21
Miscellaneous Techniques .....	9-21
Output Files .....	9-21
Output File Summary .....	9-24
Suppression of Macro Expansions .....	9-24
Functional Chart of Inline and Stand-alone Routines .....	9-25
Summary of Techniques .....	9-28

---

## Appendix A: Keywords

Toolkit Keywords .....	A-1
------------------------	-----

## Index

# Introduction

---

CA-Easytrieve Plus Toolkit enhances the capabilities of CA-Easytrieve Plus, the Computer Associates information retrieval and data management system.

CA-Easytrieve Toolkit provides easy-to-use, prewritten routines. These routines allow MIS/EDP users to further enhance the capabilities of CA-Easytrieve with minimal effort.

## Input

With CA-Easytrieve Plus Toolkit, you can retrieve any kind of record from any kind of file structure supported by CA-Easytrieve Plus, including complex data base structures such as IMS, DLI, IDMS, and SQL.

## Output

CA-Easytrieve Plus provides four types of output:

- Printed output of any kind (reports, letters, forms, mailing labels, and so forth)
- Punched card output
- File output
- Summary reports and summary output files.

## Host Language

CA-Easytrieve Plus Toolkit (referred to as Toolkit throughout this guide) is a collection of routines written in CA-Easytrieve Plus. These routines invoke CA-Easytrieve Plus macros and issue calls to routines written in other languages. Therefore, the execution of a Toolkit routine is merely the execution of a CA-Easytrieve Plus job. The basic structures that apply to Toolkit are the exact same structures that exist in CA-Easytrieve Plus.

In all Toolkit documentation, the name CA-Easytrieve Plus refers to the language that Toolkit uses to accomplish its tasks. It indicates that the document is referring to technical aspects of the host language. The name Toolkit refers to the overall characteristics of the product and to the individual routines written in CA-Easytrieve Plus.

To use Toolkit effectively you need not know CA-Easytrieve Plus. The use of Toolkit routines can be taught to personnel having no formal data processing background.

## Capabilities

Toolkit uses English-like statements to execute routines specific to MIS/EDP applications.

### Ease of Use

The basic features that make Toolkit easy to use are:

- English-language format
- Automatically formatted reports
- Boolean logic (IF, AND, OR) and relational operators (greater than, equal to, and so forth)
- File statements
- Standard numeric computation.

### Compiling Speed

Toolkit programming gives you fast preparation time and requires no compile or testing time.

## Related Publications

The following documents are available for use with CA-Easytrieve Plus Toolkit.

### CA-Easytrieve Plus

CA-Easytrieve *Programmer Guide*. Detailed information and advanced coding techniques in CA-Easytrieve.

CA-Easytrieve *Introduction to the Language*. An introduction to CA-Easytrieve, the CA-Easytrieve Plus Toolkit host language. Use this guide before you begin writing your own programs.

CA-Easytrieve *Language Reference Guide*. This guide is your source for complete system details. It contains descriptions of all product features and functions and summaries of each CA-Easytrieve version.

CA-Easytrieve Plus *Extended Reporting Facility*. Describes support of extended reporting for Impact Dot, Ink Jet, and Electro Photographic printers.

CA-Easytrieve Plus *Interface Options Guides*. These are short guides available for users of various system options. There are guides for IMS/DLI processing, CA-IDMS and IDD processing, TOTAL processing, SQL processing, CA-Datacom/DB processing, SUPRA processing, and other CA-Easytrieve Plus options.

## Command Notation

The following conventions are used throughout this guide:

<b>bold</b>	<b>Bold</b> text in program code is used to highlight an example of the use of a statement.  Text that you must enter into an input field exactly as shown is in <b>bold</b> .
{braces}	Required 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, a name, or a field used in a program example.



# Using Routines

---

This chapter describes how to use CA-Easytrieve Plus Toolkit generalized and statistical routines.

## Generalized Routines

CA-Easytrieve Plus Toolkit (Toolkit) generalized routines are designed to perform a wide array of the tasks common to general data analysis – from numerical analysis to test data generation. The routines fall into six categories:

- Integrity tests
- Date and time routines
- Numeric routines
- File comparison
- Test data generation
- Miscellaneous routines

These generalized routines can often be used with other generalized routines.

A brief description of the generalized routines, by category, is given on the following pages. A detailed description of each routine, listed in alphabetical order, is given in the chapters "[Generalized/Statistical Routines A-D](#)" and "[Generalized/Statistical Routines E-W](#)."

## Integrity Tests

In performing MIS/EDP duties, one of the major functions is to verify the validity of data in computer files. The Toolkit integrity tests assist you in performing various types of compliance tests.

The integrity test routines are both inline and stand-alone Toolkit routines. For a detailed description of how to use both inline and stand-alone routines, see the chapter "[Coding Guidelines](#)."

The integrity test routines are:

DATEVAL	Tests if the content of a specified date field is valid for a given date format. Sets a flag that reports the results of the test.
DUPTTEST	Tests a field to determine if the content appears in more than one record and reports on the results.
FLDVALR	Tests a field for a range of values and sets a flag to report on the result of the test. Reports on the results and, optionally, writes to an output file.
FLDVALT	Tests a field for values found in a table and sets a flag to report on the result of the test. Reports on the results and, optionally, writes to an output file.
FLDVALV	Tests a field for a specific value and sets a flag to report on the result of the test. Reports on the results and, optionally, writes to an output file.
GAPCHCK	Tests a numeric field to determine if records in a continuous sequence are missing from the file. Produces a report of the missing numbers.
MULTDUP	Compares up to 50 fields to determine if duplicate records exist. Optionally, writes duplicate records to an output file.
NUMTEST	Tests if the content of a specified field is numeric. Prints hexadecimal representations of non-numeric data.

## Date and Time Routines

You can easily perform date and time conversions and computations by using one of the Toolkit date and time routines. Some routines require you to define a field to hold the result of the computation or require you to interrogate a previously defined field to test the results of a computation.

All date and time routines are inline Toolkit routines. For a detailed description of how to use inline routines, see the chapter "[Advanced Techniques](#)."

Date and time routines become useful when used with other Toolkit routines. For example, you might want to produce a report of customers who took over 60 days to pay an invoice. This is accomplished using the DAYSCALC routine and one of the distribution analysis routines.

The date and time routines are:

DATECALC	Adds or subtracts a given number of days from the date in a specified field and writes the resulting date to a second field.
DATECONV	Reformats a date from a specified format to another format (for example, Gregorian to Julian, Julian to Gregorian).

DAYSAGO/DAYSAGOL	Calculates the number of elapsed days between the current date and the date in a specified field. The result is compared to a specified value, and an indicator is set if a specified criterion is met (for example, EQ, GR, GQ, LS, LQ, NQ).
DAYSCALC	Calculates the number of elapsed days between two dates. The result is written to a specified field.
GETDATE/GETDATEL	Gets the current date from the system and places it in a specified field.
TIMECONV	Converts time from hundredths of a second to hours, minutes, seconds, and hundredths of seconds.
WEEKDAY	Calculates the day of the week for a given date and positions it in a field for further use.

## Numeric Routines

The numeric Toolkit routines perform a variety of numerical calculations. All numeric routines are inline Toolkit routines. For a detailed description of how to use inline routines, see the chapter "[Advanced Techniques](#)."

The numeric routines are:

DIVIDE	Calculates the quotient and remainder of the division of two integers.
EXPO	Calculates the exponentiation of numeric values.
RANDOM	Produces a series of random numbers from 1 to 15 digits in length.
RANDSPAN	Produces a series of random numbers in a specified range.
SQRT	Calculates the square root of a number to an accuracy of two decimal places.
STDDEV	Calculates the standard deviation of a set of variables for designated intervals in the file and for the entire file.

## File Comparison

The file comparison routines are used to compare numeric data, source code, or any other form of data. They are stand-alone CA-PanAudit Plus routines. For a detailed description of how to use stand-alone routines, see the chapter "[Coding Guidelines](#)."

The file comparison routines are:

FILECOMP	Compares specified areas of records in two files and produces a report of mismatched records. Contains a method for realigning the files when a mismatch occurs. Use the facility to compare any type of data.
SRCECOMP	Compares two versions of a source program and prints a report of all added, deleted, and changed statements.

### File Compare Facility

Use the file compare facility to ensure the integrity of two files by comparing selected fields. You can compare source code, object modules, and data files.

You can make comparisons on a strict record-by-record basis or realign files using designated keys (see following explanation). When a mismatch occurs between a record pair in the primary and secondary files, both records are printed in hexadecimal, along with their character representation.

### File Compare Keys

A key is a portion of the record for which a match is searched. You can reposition the files after an unequal record pair causes a mismatch. Any field in the input record can be used as a key.

You can designate from one to six keys. If used, keys must be specified for both the primary and secondary files. They must be defined in the library section of each file and are specified in the PRIKEYS and SECKEYS parameters of the FILECOMP routine. For additional information, see the chapters "[Generalized/Statistical Routines A-D](#)" and "[Generalized/Statistical Routines E-W](#)."

When choosing keys, use alphanumeric or numeric fields where the content is in ascending sequence, for example, an alphabetic name field in a data file or the statement number field in source code.

You must specify keys in pairs (for example, for each key specified in PRIKEYS, you must specify an associated key in SECKEYS). If multiple keys are used, all key fields must match before the files are realigned.

Specification of keys is optional. However, if no key is specified, automatic repositioning cannot be performed. The files are compared on a record-by-record basis. This means if a mismatch occurs because of an unequal record pair, the remainder of the file is interpreted as being mismatched, unless records become realigned by chance.

## Examples

The two examples that follow show how files are compared on a record-by-record basis (no keys specified) and when keys are specified for the comparison.

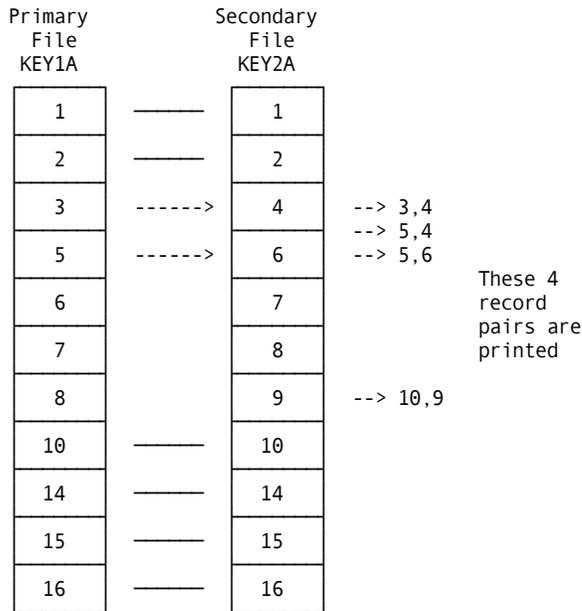
### File Compare Process Without Keys

Primary File		Secondary File		
1	———	1		
2	———	2		
3	-----	4	--> 3,4	
5	-----	6	--> 5,6	
6	-----	7	--> 6,7	These 5 record pairs are printed
7	-----	8	--> 7,8	
8	-----	9	--> 8,9	
10	———	10		
14	———	14		
15	———	15		
16	———	16		

This simplified example depicts the file compare process when no key is specified.

Comparison is made on a record-by-record basis, and no automatic repositioning takes place. In this example, record pair (3,4) causes a mismatch for the next five record pairs. The files become realigned with record pair (10,10).

File Compare Process With Keys



In this example, one pair of keys (KEY1A, KEY2A) is defined. The first mismatch occurs between record pair (3,4). When an attempt is made to realign the files, record pairs (5,4) and (5,6) are also mismatched. Alignment is achieved again by matching the key fields in record pair (6,6). Another mismatch occurs between (10,9), and the files are aligned again at (10,10).

For simplicity, this example illustrates the remainder of the record matches when the key fields match. However, even when keys match, if the specified comparison fields in the two records are not exactly identical, the record pair is diagnosed as mismatched, and printed.

**File Definitions Required**

You must code two FILE statements before invoking FILECOMP – one for the primary file and one for the secondary file. You must also define the optional key fields at this time.

The following is an example of FILE statements for the FILECOMP routine:

```
FILE PRIMARY
  PRIM-ACCT-NUM    22  4  N
  PRIM-NAME       76 20  A
  ...
FILE SECONDARY
  SEC-ACCT-NUM    22  4  N
  SEC-NAME       76 20  A
  ...
```

In this example, two pairs of keys (PRIM-ACCT-NUM, SEC-ACCT-NUM and PRIM-NAME, SEC-NAME) are defined. The first pair are numeric and begin in position 22 for a length of 4; the second pair are alphanumeric and begin in position 76 for a length of 20.

## Test Data Generation

Toolkit gives you the ability to generate specified data (alphanumeric, invalid, dates, and numeric) in user-defined formats that can be used for system tests in place of actual production data.

Each routine is an inline Toolkit routine. For a detailed description of how to use inline routines, see the chapter “[Advanced Techniques](#).” However, to successfully generate test data, you use the routines as a system—starting with one invocation of FILEGEN followed by multiple invocations of ALPHAGEN, BADGEN, DATEGEN, or NUMGEN.

## Test Data Generator Facility

The test data generator facility creates an output file and generates data in a user-defined format. Files generated with this facility can be used in lieu of existing files that might contain sensitive data. You can generate test files with characteristics identical to production files. Use these files to test routines or as an aid in learning how to use the Toolkit system.

### File Generation

The test data generator consists of the FILEGEN routine and four data generation routines:

- ALPHAGEN
- BADGEN
- DATEGEN
- NUMGEN

The FILEGEN routine establishes the output file name, the number of records, the seed for the random number generator, and whether a hexadecimal listing of output records is created. After the output file is defined with FILEGEN, use the data generation routines to create data in specified fields in the output records.

### Data Generation

Routines to generate data are invoked after the FILEGEN routine. You can invoke each of these routines any number of times and in any combination, to generate data for fields in a file. During execution, each data generation routine writes the data it produces into the field specified in the field name parameter. This process is repeated until data is generated for each record being created. The number parameter of the FILEGEN routine determines the number of test records generated.

Many options exist for the generation of data. Generated data for ALPHAGEN, NUMGEN, and DATEGEN (with the exception noted for DATEGEN) can optionally be:

- Random in a specified range
- Sequenced in increasing or decreasing order
- Incremented by a specified amount
- A series of constants (disallowed in DATEGEN)

A parameter that establishes conditions (such as the minimum or maximum for a range of random values) can either be an actual value or the name of a field containing the value.

The test data generation routines are:

ALPHAGEN	Generates alphanumeric data.
BADGEN	Generates a specified percentage of invalid data.
DATEGEN	Generates dates in any user-defined format.
FILEGEN	Defines the output file to which the data created by the other test data generation routines is written.
NUMGEN	Generates numeric data.

### Miscellaneous Routines

Toolkit provides seven miscellaneous routines that do not fit in the preceding categories of generalized routines. The miscellaneous routines are both inline and stand-alone Toolkit routines. For a detailed description of how to use both inline and stand-alone routines, see the chapter "[Advanced Techniques](#)."

The miscellaneous routines are:

ADDRCMP	Reduces addresses to a common key and produces a report of potentially duplicate addresses within the same ZIP code.
---------	----------------------------------------------------------------------------------------------------------------------

ALPHACON	Converts an edited numeric field (an alphanumeric field) into a numeric field.
APR	Calculates the annual percentage rate for a given amount and interest rate.
CONVAE/CONVEA	Convert ASCII alphanumeric data to EBCDIC and conversely.
EACHNTH	Selects every nth record (as identified by the skip factor) until it reaches the end of the input file. This interval sampling routine accepts a starting position and a skip factor.
UNBYTE	Converts a one-byte input field into eight one-byte fields, which correspond to the eight bits of the input byte.



## Coding Guidelines

---

This chapter contains general instructions and guidelines for using the CA-Easytrieve Plus Toolkit (Toolkit) system.

This chapter is an instructional module of special importance to new users of Toolkit and persons having no experience with CA-Easytrieve Plus, the host language. All material presented later in this guide presupposes a knowledge of the fundamentals given in this chapter.

The following topics are included:

- Types of Toolkit Routines (Inline and Stand-alone)
- Screening of Input Data With the Stand-alone Routines

### Types of Toolkit Routines

There are two basic types of Toolkit routines:

- Inline
- Stand-alone

These two types of routines are used differently in Toolkit and require you to supply different CA-Easytrieve Plus statements.

Overall differences between the routine types are listed in the following table and discussed in the topics that follow. In general, inline routines function in many different ways, while stand-alone routines are more consistent in their format. The following guidelines do not apply to the SMF routines, the graphing facility, or the Job Information Facility (JIF).

<b>Inline</b>	<b>Stand-alone</b>
Only one macro name is required to invoke a routine.	Generally requires two macros to invoke to invoke one routine.*
An open routine that can be combined with other routines.	A complete system that defines input files, performs the algorithm, and produces a report.
You must code a JOB statement and any required report subactivity.**	You must supply the library section.
Parameter list never contains an input file.	Parameter list always contains an input file.
In some cases you must test internally defined fields to determine action or whether to print a line of a report; in other cases, results of calculations are placed in a user-supplied field.	Creates some form of report, and in some cases an output file.

\* There are exceptions. Some stand-alone routines operate with one invocation only, and some require three macros. For further information, see the detailed descriptions for each specific routine in later chapters.

\*\* Report subactivities are discussed in the *CA-Easytrieve Plus Reference Guide*.

## Inline Routines

Inline Toolkit routines require only one macro invocation. They require varying degrees of user coding. This chapter explains the code that is standard for all inline routines. The documentation of the individual inline routines gives the specific coding you must supply.

This individual coding usually consists of testing a specially named internal field for a YES or NO condition. Based on the value in this internal field, appropriate action is then taken.

Inline routines contain the following sections:

- Environment section (optional and rarely required)
- Library section (optional)
- JOB statement (required)
- Optional user code
- Toolkit routine
- Optional user code

See the CA-Easytrieve *Language Reference Guide* for a discussion of the environment, library, and JOB sections. Aspects of the library section specific to inline routines are found next, together with an explanation of the optional user codes.

## Library Section

If your program does not use an input file, you can define working storage fields in the activity section of the job and omit the library section. However, if an input file is required, you must code the library section in your program.

## Optional User Code (Prior to Routine)

The optional user code, prior to invoking a Toolkit routine, consists of any code you want to include before invoking the routine. This can consist of field definitions, other Toolkit routines, user-defined macros, and CA-Easytrieve Plus statements.

You must be careful to follow CA-Easytrieve Plus coding guidelines for any statements coded in this section. For example, report subactivities are not allowed in this section.

## Toolkit Routine

You invoke an inline Toolkit routine as you would a macro, coding a percent sign (%) followed by the name of the routine and any applicable parameters as shown in the following example:

```
%routine-name parm1 parm2 ... parmn
```

Routine names and syntax of this statement are fully described in the chapters [“Generalized/Statistical Routines A-D”](#) and [“Generalized/Statistical Routines E-W.”](#)

## Optional User Code (Following Routine)

This optional section consists of any code you want to include after the Toolkit routine is finished. It can consist of testing internal flags of the routine, another Toolkit routine, or any valid CA-Easytrieve Plus statements. You must be careful to follow all CA-Easytrieve Plus coding guidelines for any statements coded in this section.

If an inline routine is coded using the JOB INPUT NULL statement, a STOP statement must be coded to terminate execution of the job. For details regarding the JOB statement and inhibiting automatic input, see the description of the JOB statement in the CA-Easytrieve Plus *Reference Guide*.

## Stand-alone Routines

Stand-alone Toolkit routines are complete systems in that they require an input file, perform specific processing, and generate a listing and possibly an output file. Stand-alone routines are more structured than inline routines and follow a standard format. However, in this standard format, there are two distinct types of stand-alone routines:

- Logic-before-invocation
- Logic-after-invocation

Each type uses different types of CA-Easytrieve Plus processing. You can invoke both types of stand-alone routines in an identical manner in most applications. In some sophisticated applications, you may want to use the processing capabilities of both types of stand-alone routines. For additional information about logic-before-invocation and logic-after-invocation, see the chapter "[Advanced Techniques](#)."

## Stand-alone DISPLAY and Stand-alone REPORT Routines

The two types of stand-alone routines are:

- DISPLAY
- REPORT

Their names come from the CA-Easytrieve Plus statements that print listings. In CA-Easytrieve Plus, a DISPLAY statement prints lines of output but does not perform any special functions. The REPORT statement prints lines of output and can perform automatic page numbering, titling, footing, control breaks, summing, and other special functions.

This guide uses the term *stand-alone* to denote both DISPLAY and REPORT type routines and that the information applies to both types. If the information is specific to one of the types, then the guide will use the terms *stand-alone DISPLAY* and *stand-alone REPORT*.

The following table shows the different places in your programs and routines where you must code these two types of output statements:

<b>DISPLAY Statements</b>	<b>REPORT Statements</b>
You must code DISPLAY statements as inline statements.	A REPORT statement is a subactivity and must appear at the end of a program.
Display statements can appear almost anywhere in a Toolkit program.	A stand-alone REPORT routine is, in effect, closed off by the existence of the REPORT statement at the end of the routine.
A stand-alone DISPLAY routine is open in nature, and you can code additional statements after a stand-alone DISPLAY routine.	The type of code that can occur after the report statement is limited.

Consult the CA-Easytrieve Plus *Reference Guide* for details regarding the DISPLAY and REPORT statements.

**Note:** The chapter “[Advanced Techniques](#)” explains the different processing options for DISPLAY and REPORT stand-alone routines and contains examples illustrating how you can use each type.

### Stand-alone Routine Format

The structure of a stand-alone routine differs from the structure of an inline routine; therefore, the two types of routines have different mandatory and optional sections. In a stand-alone routine, you must specify the library section and the Toolkit routine (first invocation). There are also optional sections in a stand-alone routine.

Stand-alone routines contain the following sections:

- Environment section (optional and rarely used)
- Library section (required)
- Logic-before-invocation (optional)
- Toolkit routine (first invocation, required)
- Screening logic (optional)
- Toolkit routine (second invocation - if required)
- Logic-After-Invocation (optional)

## Library Section

All stand-alone routines require a library section. The library section must describe the input and optional output files that the macro invocation specifies. You can define working storage fields either in the library section or in the activity section of your program (see [Screening Logic](#) later in this chapter).

## Logic-Before-Invocation

The logic before the invocation of the first macro must consist of an entire CA-Easytrieve Plus activity or group of activities. This includes:

- JOB activities
- SORT activities
- REPORT subactivities

Use an activity defined in this section to screen the input file or calculate data, such as a population size, before invoking the Toolkit routine. You can also use an activity defined in this section to print a report prior to the execution of the routine.

The logic-before-invocation section is optional and is most often used in sophisticated applications. For details and examples of logic-before-invocation, see the chapter "[Advanced Techniques](#)."

## Toolkit Routine (First Invocation)

For most Toolkit stand-alone routines, you must invoke two macros to execute the routine. This is done to allow you to screen the input file and code any CA-Easytrieve Plus logic before the routine begins processing. This optional screening logic (discussed next) is placed between the two invocation statements for the routine.

Invocation of the first Toolkit stand-alone routine consists of a percent sign (%) followed by the name of the routine and the character 1, followed by any applicable parameters as shown in the following example:

```
%routine-name1 parm1 parm2 ... parmn
```

Routine names and syntax are fully described in the chapters "[Generalized/Statistical Routines A-D](#)" and "[Generalized/Statistical Routines E-W](#)."

## Screening Logic

One of the processing options of Toolkit is the ability to screen input data. The screening of the input file allows you to examine fields in the input record and eliminate records from processing. It also allows you to perform complex processing and decision making.

The use of the screening facility is not required for the successful execution of any Toolkit stand-alone routine.

Complex processing requires an in-depth knowledge of CA-Easytrieve Plus and is not discussed in detail here. [Example Two - With Screening Logic](#), later in this chapter, demonstrates an example of complex screening logic.

Screening logic is coded between the macro invocation statements. You code screening logic after the first invocation of the macro.

For example, if you want records bypassed from processing, you must do the following:

- Code the invocation of the first macro.
- Code your screening logic and use the CA-Easytrieve Plus branching statement GO TO JOB.

This causes control to be passed to the JOB statement which, in effect, bypasses the record from processing and reads the next record from the input file.

- Code the invocation of the second macro.

[Example Two - With Screening Logic](#), later in this chapter, contains screening logic.

Some stand-alone routines do not allow screening. This is because unpredictable results may occur in these routines if screening processing is allowed. Any routine that requires the invocation of two macros will allow input file screening. Stand-alone routines that require only one macro invocation do not allow screening.

## Toolkit Routine (Second Invocation)

The second invocation of the routine consists of a percent sign (%) followed by the name of the routine and the character 2, and usually no parameters as shown in the following sample. The only parameters specified on the second invocation are the options that control output files and report subactivities.

```
%routine-name2 [parm1 parm2 ... parmn]
```

The second invocation is required for all stand-alone routines that allow screening of input files.

## Logic-After-Invocation

Logic-after-invocation is an optional section of code that sophisticated applications often use. Logic-after-invocation follows the final macro of a stand-alone routine and can consist of different CA-Easytrieve Plus statements depending on whether the routine is a DISPLAY or REPORT type.

The following example is a DISPLAY type:

```
FILE . . . .  
FILE . . . .  
  
Stand-alone routine . . . .  
(language statements)  
(screening code)  
DISPLAY . . . .
```

The following example is a REPORT type:

```
FILE . . . .  
FILE . . . .  
  
Stand-alone routine . . . .  
  
REPORT . . . .
```

The logic coded in this section most often consists of statements that the screening code section performs. For details and examples of both types of stand-alone routines (DISPLAY and REPORT), see the chapter "[Advanced Techniques](#)."

You can also use logic-after-invocation for REPORT procedures. For details and an example of this application of stand-alone routines (DISPLAY and REPORT), see the chapter "[Advanced Techniques](#)."

## Examples

The following are examples of stand-alone Toolkit routines.

### Example One - Without Screening Logic

```
FILE CUSTFIL  
BALANCE 1 10 P 2  
MONTH 11 2 N  
%VERSUS1 CUSTFIL BALANCE MONTH GRAPH 0 2  
%VERSUS2
```

This example demonstrates the use of a stand-alone routine. It contains the mandatory library section for the input file, CUSTFIL. The VERSUS routine contains two macros, but no screening logic is performed.

**Note:** The second macro has no parameters.

In this particular example, the distribution analysis routine VERSUS is used to show the frequency distribution of values from the field BALANCE in a specified month. The user has asked for a graph as part of the output report.

### Example Two - With Screening Logic

```
FILE CUSTFIL
  BALANCE   1 10 P 2
  MONTH    11 2 N
%VERSUS1 CUSTFIL BALANCE MONTH GRAPH 0 2
IF BALANCE LE 0
  GO TO JOB
END-IF
%VERSUS2
```

This example is identical to the first except that logic to screen input records is included between the invocations of VERSUS1 and VERSUS2. VERSUS performs its calculations properly whether screening logic is present or not.

The effect of the screening logic in this example is to bypass all records with a balance less than or equal to zero. This causes the output produced by VERSUS to be a report of only positive balances.

Generally, screening logic consists of three statements:

- An IF statement
- A GO TO JOB statement
- An END-IF statement

If the END-IF statement is missing, a CA-Easytrieve Plus syntax error is detected, and a diagnostic message is printed. For a further explanation of this message, see the appendix "Diagnostics" in the CA-Easytrieve Plus *Reference Guide*.



# Generalized/Statistical Routines

## A-D

This chapter lists alphabetically, and gives detailed descriptions of, routines ADDR\_CMP through DUPTEST.

### ADDR\_CMP

The ADDR\_CMP routine produces a report of potentially duplicate street addresses in a boundary file, usually defined as the ZIP code. Standard keys are created for each address to account for different representations of the same address. For example, P.O. Box #54 and POST OFFICE BOX 54 have the same standard key of POBOX54, and are flagged by ADDR\_CMP as duplicate addresses. A boundary field is defined to eliminate the flagging of duplicate addresses in different areas, such as cities, states, or ZIP codes.

### Syntax

```
%ADDR_CMP1 infile [LRECL length]
%ADDR_CMP2 infile field boundary {outfile} {field2 field3}
{NOFILE } {SUMMARY }
```

infile

Specify the name of the input file to ADDR\_CMP. A valid name is any previously defined file.

[LRECL length]

Optionally specify the length of the input record. The default is 32,767 bytes. If the record length is less than 32,767, you can improve the efficiency of disk storage utilization and execution speed by specifying the exact length of the record using the following formula:

$$\text{infile-lrecl} + 39 \text{ work bytes} + 4 \text{ RDW bytes} = \text{LRECL}$$

field

Specify the field for which duplicate street addresses are searched. If the address in this field is the same in two or more records, a duplicate exists. A valid name is any field in infile.

boundary

Specify the boundary field in the infile input file for duplicate address checking. This is usually the ZIP code field. This causes ADDR\_CMP to check for duplicate addresses only in this boundary and eliminates the flagging of duplicate addresses in other areas.

{outfile}  
{NOFILE }

Specify whether records with duplicate addresses are to be written to an output file:

**outfile**—Specify the file to which the duplicate records selected are written. File characteristics must be coded on the FILE statement for this output file. The file specified on outfile must have the same file characteristics as the input file. Valid names for outfile include any previously defined file. If you want to include the standard address key created by ADDR\_CMP, increase the outfile size by 38 and add the following field definition to the outfile definition:

```
ADDRCMP-KEY  start  38  A
```

where start is the length of infile plus one.

The standard address key is placed in the ADDR\_CMP-KEY field.

**NOFILE**—Specify this option if records with duplicate addresses are not to be written to an output file.

{field2 field3}  
{SUMMARY }

Specify whether the report contains a detail line for each duplicate record or is a summary report giving the total of duplicates.

**field2 field3**—When this option is specified, three fields are listed for each duplicate record:

- Field (the field for which duplicates are searched)
- Field2
- Field3

Valid names are any previously defined fields.

**SUMMARY**—When this option is specified, a summary report is produced. This consists of the total number of sets of duplicates and the total of all duplicate records in the file.

---

## Operation - Stand-alone REPORT

Use the ADDRCPM routine to get information on duplicate street addresses or a summary of duplicate street addresses.

## Operation - Database

ADDRCPM can access both database and nondatabase files without changes in specification parameters. The infile parameter can be a nondatabase file name or the name of a database file defined in the library section.

## Examples

The following are two examples of the ADDRCPM routine.

### Example One

In this example, NAME and ACCNT are used for ID fields, and the detail report is specified.

Input

```
FILE ADDRFILE F(80)
  NAME      1 18 A
  ADDRESS   19 30 A
  CITY      49  5 A
  STATE     54  2 A
  ZIP       56  5 N
  ACCNT     62  8 N
%ADDRCMP1 ADDRFILE
%ADDRCMP2 ADDRFILE ADDRESS ZIP NOFILE NAME ACCNT
```

Output

REPORT OF DUPLICATE ADDRESSES  
 INPUT FILE: ADDRFILE BOUNDARY: ZIP FIELD: ADDRESS  
 NOFILE IS PRODUCED

ZIP	NORMALIZED KEY	ACTUAL ADDRESS	NAME	ACCNT	NUMBER OF DUPLICATES
11111	POBOX111	P O BOX111	JACK DOE	00066666	
		P O BOX 111	JOSEPH KOOL	00000000	
		P. O. BOX 111	PETER GUNN	00077777	
		P.O. BOX 111	WILD BILL HAYCOCK	00011111	
		POST OFFICE BOX 111	WILLIAM SCOTT	00055555	
	ADDRCMP-KEY TOTAL				5
11111	RR2BOX69	R.F.D. 2, BOX 69	BOB JOHNS	00155554	
		R.F.D. #2 BOX 69	HERB LEWIS	00166665	
		R R 2 BOX 69	JANE ROBERTS	00099999	
		R.R. 2, BOX 69	PETER PAUL	00111110	
		RURAL ROUTE 2 BOX 69	SCOTT JONES	00122221	
		RFD 2, BOX 69	TOM PATTERSON	00133332	
		RR 2, BOX 69	WILBUR SMITH	00088888	
	ADDRCMP-KEY TOTAL				7
11111	1234SUNSHINE	1234 SUNSHINE STREET	JANE BABBITT	00188887	
		1234 SUNSHINE PLACE	JESSIE SMITH	00199998	
	ADDRCMP-KEY TOTAL				2
	ZIP TOTAL				14
22222	HCR5BOX9	HIGHWAY CONTRACT ROUTE 5 BOX 9	JOHN MARIO	00555550	
		HCR #5, BOX 9	ROBERTO NUNEZ	00577772	
	ADDRCMP-KEY TOTAL				2
22222	456NALASKA	456 N. ALASKA STREET	JANE DOE	00488884	
		456 NORTH ALASKA STREET	UUG JOHNS	00466662	
	ADDRCMP-KEY TOTAL				2
	ZIP TOTAL				4
40372	HCR5BOX9	HCR #5, BOX 9	BARNEY RUBBLE	01211160	
		HCR #5, BOX 9	MIGUEL TORRES	01090044	
		HIGHWAY CONTRACT ROUTE 5 BOX 9	PAUL SMITH	01170788	
		HCR #5, BOX 9	SAMSON JONES	01130416	
	ADDRCMP-KEY TOTAL				4
	ZIP TOTAL				4
	FINAL TOTAL				22

## Example Two

In this example, SUMMARY is specified, and an outfile is produced. A subsequent JOB activity reports the contents of the output file.

Input

```
FILE ADDRFILE F(80)
NAME      1 18 A
ADDRESS   19 30 A
CITY      49  5 A
STATE     54  2 A
ZIP       56  5 N
ACCNT     62  8 N
FILE OUTFILE F(80)
%ADDRCMP1 ADDRFILE
%ADDRCMP2 ADDRFILE ADDRESS ZIP OUTFILE SUMMARY
```

Output

```
                SUMMARY REPORT OF DUPLICATE ADDRESSES
INPUT FILE: ADDRFILE   BOUNDARY: ZIP   FIELD: ADDRESS
                OUTFILE IS PRODUCED
```

	TOTAL NUMBER OF DUPLICATES IN ZIP
ZIP TOTAL	14
ZIP TOTAL	4
ZIP TOTAL	4
FINAL TOTAL	22

## ALPHACON

The ALPHACON routine converts an edited numeric field (alphanumeric) into a numeric field, which can be used for calculations. If the input field contains alphabetic characters, they are removed by ALPHACON in the conversion. The number of decimal places in the converted number is defined in the file definition.

CA-Easytrieve Plus edit masks allow the use of any character as a negative number indicator, but ALPHACON checks only the most typical negative number indicators:

- Minus sign (-)
- Credit indicator (CR)
- Two debit indicators (DB, DR)

If one of these negative indicators is found, ALPHACON converts the indicator to a minus sign (-) and places it at the end of the converted number. If ALPHACON encounters a negative indicator that it does not recognize, ALPHACON ignores the indicator, and the number is converted to a positive number.

## Syntax

```
%ALPHACON  alphafield  numfield  [DECIMAL 'separator']
```

alphafield

Specify the alphanumeric field to be converted. The field has a maximum length of 30 characters.

numfield

Specify the numeric field to which the converted number is written. The converted number is truncated to the number of places specified in the field definition. The maximum number of digits supported is 17.

[DECIMAL 'separator']

Specify the symbol used as a separator between the integer and the decimal in the input field if it is not a decimal point (.).

## Operation - Inline

ALPHACON generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

No change in specification of parameters is required to use ALPHACON with database files.

## Examples

The following is an example of the ALPHACON routine.

### Input

In this example, ALPHACON converts the field into numeric data. ALPHACON-FLAG is set to YES if the value is converted correctly. If the field is truncated on the right, left, or on both sides, ALPHACON-FLAG is set to RIGHT, LEFT, or BOTH, respectively.

```
FILE INFILE CARD
ALPHA      1 30 A
NUMBER     W 17 N 5
*
JOB INPUT INFILE
%ALPHACON ALPHA NUMBER
PRINT ALPHACON-REPORT
REPORT ALPHACON-REPORT LINESIZE 78
TITLE 1 'ALPHANUMERIC CONVERSION'
LINE ALPHA ALPHACON-FLAG NUMBER
END
```

### Output

```
5/20/88                ALPHANUMERIC CONVERSION                PAGE      1

                ALPHA                ALPHACON-FLAG                NUMBER

1                YES                1.00000
-2.3            YES                2.30000-
456,789.0123456789012 RIGHT        456,789.01234
$***3,157.36    YES                3,157.36000
5711.837392-   RIGHT        5,711.83739-
38,351.99CR    YES                38,351.99000-
1,398,351.23DR YES                1,398,351.23000-
123.45DB      YES                123.45000-
1#800#111#2222 YES                18,001,112,222.00000
A1B2C3D4E5F6G7H8I9J0K1L2M3N LEFT        234,567,890,123.00000
```

## ALPHAGEN

The ALPHAGEN routine generates alphanumeric data and writes it to the field parameter.

**Note:** The FILEGEN routine must be invoked to use the ALPHAGEN routine.

### Syntax

```
%ALPHAGEN field 'literal' {BETWEEN minimum maximum          }  
                          {SEQUENCE from to increment         }  
                          {CONSTANT 'value1,value2, ... ,valuen'}
```

field

Specify the name of the field where ALPHAGEN places the alphanumeric data it generates. This field must be large enough to contain the characters specified by the literal parameter plus the characters specified by the BETWEEN, SEQUENCE, or CONSTANT parameter. Valid names include any alphanumeric field defined in the file name specified in FILEGEN. The field must have a data format of A (alphanumeric).

'literal'

Specify any combination of alphanumeric characters to be used as a prefix for data generated by the BETWEEN, SEQUENCE, or CONSTANT parameter. Valid literals include any sequence of 1 through 15 alphanumeric characters. Use single quotation marks around the data.

{BETWEEN minimum maximum}

The BETWEEN keyword causes random numbers to be generated in the range specified by the minimum and maximum parameters that you specify. The value is appended to the literal parameter and written to the field parameter. Valid values for minimum and maximum are actual numeric values or the name of a field containing the value. Minimum and maximum cannot be more than 15 digits. The seed for the random generation of values is obtained from the FILEGEN routine.

Values generated are greater than or equal to the minimum and less than the maximum.

{SEQUENCE from to increment}

The SEQUENCE keyword causes a fixed set of numbers to be generated. The set of numbers begins with the from value and is incremented for each record by the increment value until the to value is equaled or exceeded. The sequence is repeated beginning with the from value. Each number is appended to the literal parameter and written to the field parameter. Valid values for from, to, and increment are actual numeric values or the name of a field containing the value.

To generate a decreasing set of numbers, specify a from value greater than the to value, and code a negative increment.

{CONSTANT 'value1, value2, . . . ,valuen'}

The CONSTANT keyword generates a specified series of values. The sequence is repeated until a value has been generated for each record being created. Each number is appended to the literal parameter and written to the field parameter.

Separate the values in the constant string by commas, and enclose the string in single quotation marks. Valid values for value1 to valuen consist of actual alphanumeric values 1 through 15 characters in length. The entire length of the literal portion of CONSTANT is limited to 40 characters, including commas.

## Operation - Inline

Use the ALPHAGEN routine only after you have specified the FILEGEN routine. ALPHAGEN can be specified with DATEGEN, NUMGEN, and BADGEN. Conditional execution of ALPHAGEN is discussed in [Example 3 - Conditional Execution of Data Generation Routines](#) following the BADGEN routine.

## Operation - Database

ALPHAGEN, with FILEGEN, cannot be used in a database application.

## Examples

The following example illustrates how ALPHAGEN is used with FILEGEN:

```
FILE CENTFILE F(24)
  NAME      1 12 A
  EMP#     13  5 N
  BIRTH    18  6 N
*
JOB INPUT NULL
%FILEGEN CENTFILE 25 5 NOHEX
%ALPHAGEN NAME ' CUST ' CONSTANT ' JOHNSON, SMITH, PETERS '
%NUMGEN EMP# SEQUENCE 1 10050 1
%DATEGEN BIRTH MMDDYY 0 BETWEEN 010151 010175
```

The following examples illustrate the three uses of ALPHAGEN. An example of the full facilities of the test data generation routines is found following the BADGEN routine.

## BETWEEN

This example generates a random value between 10 and 94. Each value is appended to the literal PERSON and written to the NAME field.

```
%ALPHAGEN NAME 'PERSON' BETWEEN 10 95
```

## SEQUENCE

A value of 4 is appended to the literal PERSON and written to the NAME field of the first record. PERSON6 is written to the second record, PERSON8 to the third record, and so forth, in increments of 2 until the to value (98) is equaled or exceeded. The sequence is repeated starting with PERSON4.

```
%ALPHAGEN NAME 'PERSON' SEQUENCE 4 98 2
```

## CONSTANT

J B JOHNSON is written to the NAME field of the first record; J B SMITH is written to the second record; and J B PETERS is written to the third record. This sequence is repeated until the NAME field has been generated for each record being created.

```
%ALPHAGEN NAME 'J B' CONSTANT ' JOHNSON, SMITH, PETERS'
```

## APR

The APR routine calculates the effective annual percentage rate for a given annual percentage rate and period. The formula for calculating the effective annual percentage rate is:

$$\text{EFFAPR} = ((1 + \text{apr})/\text{period})^{**} \text{poc} - 1$$

Where:

**apr**—annual percentage rate

**period**—The number of times interest is compounded in a year. For example, if the interest rate is compounded quarterly, period = 4; if monthly, period = 12; if daily, period = 365.

**poc**—Number of periods compounded in a year. For example, to compound yearly, and the period = 12, then poc = 12. To compound quarterly, and period = 12, poc = 3.

To compute periods of continuous compounding, set period and poc to the same value.

## Syntax

```
%APR  apr  period  poc  effapr
```

apr

Specify the annual percentage rate to use. A valid value is an actual numeric value or the name of a field containing a numeric value.

period

Specify the number of times in a year the interest is compounded. A valid value is an actual numeric value or the name of a field containing a numeric value. The value of period cannot be 0. If the value of period is 0, diagnostics message PAP310 is returned, and APR terminates.

poc

Specify the number of periods of compounding in a year to use in calculating the effective annual percentage rate. A valid value is an actual numeric value or the name of a field containing a numeric value.

effapr

Specify the name of the numeric field to write the calculated effective annual percentage rate.

## Operation - Inline

APR generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

No change in specification of parameters is required to use APR with database files.

## Example

In this example, the effective APR is calculated and printed.

Input

```

FILE INFILE
PERCRATE  1 2 N HEADING ('PERCENTAGE' 'RATE')
PERD      3 2 N HEADING ('NUMBER' 'OF' 'PERIODS')
POC       5 2 N HEADING ('PERIODS' 'OF' 'COMPOUND')
*
DEFINE EFFAPR W 10 N 2 HEADING ('ANNUAL' 'PERCENTAGE' 'RATE')
*
JOB INPUT INFILE
  %APR PERCRATE PERD POC EFFAPR
  PRINT APRDATA
*
REPORT APRDATA LINESIZE 78
  TITLE 'APR TEST DATA EXAMPLES'
  LINE PERCRATE PERD POC EFFAPR

```

Output

```

4/13/88                APR TEST DATA EXAMPLES                PAGE      1

```

PERCENTAGE RATE	NUMBER OF PERIODS	PERIODS OF COMPOUND	ANNUAL PERCENTAGE RATE
01	01	12	12.68
02	02	12	12.68
03	03	12	12.68
04	14	12	3.48
05	05	12	12.68
06	01	12	101.21
07	12	12	7.22
07	12	24	14.98
07	12	01	.58
08	23	12	4.25
09	04	12	30.60
10	14	12	8.91
11	25	22	10.14
12	06	12	26.82
13	17	12	9.57
14	08	12	23.14
15	09	12	21.93
16	10	12	20.98
17	11	12	20.20
18	12	12	19.56

---

## BADGEN

The BADGEN routine generates invalid data by overlaying data previously created by one of the data generation routines. You must invoke the appropriate routine (ALPHAGEN, DATEGEN, or NUMGEN) before invoking BADGEN, so that a particular field can be overlaid with invalid data.

For numeric fields, BADGEN generates non-numeric data; for alphanumeric fields, BADGEN generates non-alphanumeric characters. Invalid data is generated randomly for the specified percentage of the file.

### Syntax

```
%BADGEN field percent [RSGROUP number]
```

field

Specify the name of the field that you want BADGEN to overlay with invalid data. The name must be the same as the field parameter specified on a previous data generation routine.

percent

Specify a numeric value between 1 and 99 that represents a percent of all records being created (as established by the FILEGEN routine). This parameter represents the percent of records that will be generated with invalid data.

[RSGROUP number]

This optional parameter specifies a numeric value that starts at 1 and is incremented by 1 for each BADGEN coded in a FILEGEN. The default value is 1. If only one BADGEN is used, this parameter can be omitted.

### Operation - Inline

You must invoke BADGEN after the data-generation routine for which it is creating invalid data.

### Operation - Database

BADGEN, used with FILEGEN, ALPHAGEN, DATEGEN, and NUMGEN, cannot be used in a database application.

## Examples

This section provides examples of the BADGEN routine.

### Example 1 - Generating Random Values

In this example, NUMGEN generates a random value between 3 and 97 for the ZONE field. BADGEN overlays 15 percent of these values with non-numeric data.

```
FILE ...
  ZONE      1  5  N
  ...
%NUMGEN ZONE BETWEEN 3 98
%BADGEN ZONE 15
```

### Example 2 - Generating Calculated Test Data

In addition to using NUMGEN, you can generate numeric test data by means of a calculation. Additionally, the calculation can be preceded by conditional logic that states the conditions under which the statement or statements are executed. The following example demonstrates data generated with conditional logic and calculations:

```
FILE PAYFILE ...
  GROSS-PAY ...
  TAX-CLASS ...
  DEDUCTIONS ...
  BIRTH ...
JOB INPUT NULL
%FILEGEN PAYFILE 200 13243 HEX
%NUMGEN GROSS-PAY BETWEEN 200 1000
%NUMGEN TAX-CLASS CONSTANT '2,3,1,3,1,2'
IF TAX-CLASS = 1
  DEDUCTIONS = GROSS-PAY * .1
END-IF
IF TAX-CLASS = 2
  DEDUCTIONS = GROSS-PAY * .15
END-IF
IF TAX-CLASS = 3
  DEDUCTIONS = GROSS-PAY * .18
END-IF
%DATEGEN BIRTH MMDDYY BETWEEN 010130 123165
```

This example contains a series of formulas that calculate the value for DEDUCTIONS. Each formula is preceded by an IF statement containing the TAX-CLASS field generated by NUMGEN to determine which formula is used.

**Note:** A value is generated for both the GROSS-PAY and TAX-CLASS fields prior to the logic that determines the value for DEDUCTIONS.

### Example 3 - Conditional Execution of Data Generation Routines

Any data generation routine can be conditionally executed by preceding the invocation of the routine with an IF statement and following it with an END-IF statement. This allows data to be generated for a field based on the decision logic. You can have multiple invocations of the data generation routines for the same field.

To facilitate the conditional generation of data, the FILEGEN routine maintains two fields:

**FILE-COUNT** – The number of the record currently being created.

**USER-RANDOM** – Random number between 0.0 and 99.9.

The following example demonstrates the conditional execution of data generation routines:

```
FILE PAYFILE ...
  GROSS-PAY ...
  TAX-CLASS ...
  BIRTH ...
JOB INPUT NULL
%FILEGEN PAYFILE 200 13243 HEX
%NUMGEN TAX-CLASS CONSTANT '2,3,1,3,1,2'
IF FILE-COUNT LE 150
  %NUMGEN GROSS-PAY BETWEEN 200 500
END-IF
IF FILE-COUNT GT 150
  %NUMGEN GROSS-PAY SEQUENCE 500 1000 50
END-IF
%DATEGEN BIRTH MMDDYY BETWEEN 010130 123165
```

In this example, the NUMGEN routine is coded twice for the GROSS-PAY field. For records 1 through 150, the NUMGEN routine generates a random GROSS-PAY value of between 200 and 499. For records 151 to 200, NUMGEN generates a fixed set of numbers beginning with 500, and incrementing the value by 50 until 1000 is reached. This sequence is repeated until a GROSS-PAY value is generated for all 200 records.

### Example 4 - Generating Invalid Data

You can generate invalid data for a field by one of two methods:

- Use BADGEN to overlay a field with bad data.
- Code a routine to generate invalid data. This method is demonstrated in the following example.

```
FILE PAYFILE ...
  REGION ...
  ZONE ...
JOB INPUT NULL
%FILEGEN PAYFILE 200 13243 HEX
IF USER-RANDOM LT 90
  %NUMGEN REGION BETWEEN 1 10
ELSE
```

```
%NUMGEN REGION BETWEEN 10 101
END-IF
IF USER-RANDOM LT 30
  %NUMGEN ZONE CONSTANT '1,2,3'
END-IF
IF USER-RANDOM EQ 30 THRU 70
  %NUMGEN ZONE CONSTANT '4,5,6,7,8'
END-IF
IF USER-RANDOM GT 70
  %NUMGEN ZONE CONSTANT '12,15,17'
END-IF
```

Assume that the valid values for REGION are the numbers 1 through 9. This example generates valid values in approximately 90 percent of the records. The remaining records contain invalid values between 10 and 100.

This example also illustrates how to use USER-RANDOM to control the approximate distribution of values. Assuming that valid values for ZONE are the numbers 1 through 8, the conditional logic generates valid ZONEs for approximately 70 percent of the records. In addition, approximately 30 percent contain the values 1, 2, or 3, and 40 percent contain 4, 5, 6, 7, or 8. The remaining records contain the invalid values 12, 15, or 17.

## CONVAE

The CONVAE routine converts ASCII alphanumeric characters to their EBCDIC equivalent. CONVAE is an inline routine that you can use with other Toolkit routines or CA-Easytrieve Plus logic. The inline design of CONVAE reduces execution time and reduces requirements for temporary or permanent storage space. You can also use CONVAE to create a permanent file of converted data.

For conversion from EBCDIC to ASCII, see the [CONVEA](#) routine.

### Syntax

```
%CONVAE [DBFILE] [STARTPOS identifier] [LENGTH value]
```

[DBFILE]

Specify this optional parameter only for database use of CONVAE. Specify only the literal DBFILE, not the name of the active input file.

[STARTPOS identifier]

This optional parameter specifies the starting position for the conversion process. The identifier must be a previously defined field.

Conversion takes place in the active input file starting at the position defined by STARTPOS and continuing for the length specified by the length parameter.

The default value for STARTPOS is the first byte of the active input file. This means that when using the default value the conversion begins with the first byte of the active input file.

The requirements for specifying STARTPOS are different when you use CONVAE in a database application. In this case the STARTPOS parameter is no longer optional – it is a required parameter.

[LENGTH value]

This optional parameter specifies the length, or number of bytes, that you want to convert. The conversion starts at the position that STARTPOS defines and continues for the length that the LENGTH parameter specifies. The default value is the record length of the current record. A valid value is an actual numeric value or the name of a field containing a numeric value. The value that you specify for LENGTH plus the numeric value of STARTPOS must be less than or equal to the length of the current record.

## Operation - Inline

STARTPOS and LENGTH are optional parameters. If you want to convert the entire active input file, do not specify STARTPOS and LENGTH. STARTPOS and LENGTH default to values that convert the active input file starting at the first byte and continuing for the length of the input record.

This is extremely useful for variable length files. Since each record may have a different length, allowing these parameters to default assures the conversion of all bytes of all records. Bytes that cannot be converted are changed to hexadecimal '07'.

CONVAE can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

The DBFILE parameter identifies CONVAE as a routine that can access database files. This parameter is optional, and you do not have to specify it for nondatabase use. However, you must specify this parameter when using CONVAE in a database application.

**Note:** The specification of the STARTPOS parameter will differ when you use CONVAE in a database application.

## Example

Following is an example of CONVAE.

Input

```
FILE ASCII
  field-name ...
...
FILE EBCDIC ...
  field-name ...
...
JOB INPUT ASCII
%CONVAE
PUT EBCDIC FROM ASCII
```

Results

This example demonstrates the conversion of a file from ASCII to EBCDIC and the subsequent creation of the converted file. The JOB INPUT ASCII statement defines the active input file. Each record is read from the ASCII file and all bytes are converted by the CONVAE routine. The PUT statement writes the converted data to the EBCDIC file.

## CONVEA

The CONVEA routine converts EBCDIC alphanumeric characters to their ASCII equivalent. CONVEA is an inline routine that you can use with other Toolkit routines or CA-Easytrieve Plus logic. The inline design of CONVEA reduces execution time and reduces requirements for temporary or permanent storage space. You can also use CONVEA to create a permanent file of converted data.

For conversion from ASCII to EBCDIC, see the [CONVAE](#) routine.

## Syntax

```
%CONVEA [DBFILE] [STARTPOS identifier] [LENGTH value]
```

[DBFILE]

Specify this optional parameter only for database use of CONVEA. Specify only the literal DBFILE, not the name of the active input file.

[STARTPOS identifier]

This optional parameter specifies the starting position for the conversion process. The identifier must be a previously defined field.

Conversion takes place in the active input file starting at the position defined by STARTPOS and continuing for the length specified by the length parameter.

The default value for STARTPOS is the first byte of the active input file. This means that when using the default value the conversion begins with the first byte of the active input file.

The requirements for specifying STARTPOS are different when you use CONVEA in a database application. In this case the STARTPOS parameter is no longer optional – it is a required parameter.

[LENGTH value]

This optional parameter specifies the length, or number of bytes, that you want to convert. The conversion starts at the position that STARTPOS defines and continues for the length that the LENGTH parameter specifies. The default value is the record length of the current record. A valid value is an actual numeric value or the name of a field containing a numeric value. The value that you specify for LENGTH plus the numeric value of STARTPOS must be less than or equal to the length of the current record.

## Operation - Inline

STARTPOS and LENGTH are optional parameters. If you want to convert the entire active input file, do not specify STARTPOS and LENGTH. STARTPOS and LENGTH default to values that convert the input file starting at the first byte and continuing for the entire length of the input record.

This is extremely useful for variable length files. Since each record may have a different length, allowing these parameters to default assures the conversion of all bytes of all records. Bytes that cannot be converted are changed to hexadecimal '7F'.

CONVAE can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

The DBFILE parameter identifies CONVEA as a routine that can access database files. This parameter is optional, and you do not have to specify it for non-database use. However, you must specify this parameter when using CONVEA in a database application.

**Note:** The specification of the STARTPOS parameter will differ when you use CONVEA in a database application.

## Example

The following is an example of CONVEA.

Input

```
FILE EBCDIC ...
  field-name ...
  ...
FILE ASCII ...
  field-name ...
  ...
JOB INPUT EBCDIC
%CONVEA
PUT ASCII FROM EBCDIC
```

Results

This example demonstrates the conversion of a file from EBCDIC to ASCII and the subsequent creation of the converted file. The JOB INPUT EBCDIC statement defines the active input file. Each record is read from the EBCDIC file and is converted by the CONVEA routine. The PUT statement writes the converted data to the ASCII file.

## DATECALC

The DATECALC routine adds or subtracts a given number of days from the date specified in a field and writes the resulting date to a second field.

## Syntax

```
%DATECALC date1 format1 {PLUS } days date2 format2 [THRESHOLD value]
                    {MINUS}
```

date1

Specify the name of the field containing the date to which a given number of days are to be added or subtracted. The date in this field must be in the format specified by format1. A valid name is any previously defined field.

format1

Specify the format of the date1 field. This is a literal description of pairs of letters. The letters indicate positions as follows:

MM = month  
DD = day  
YY = year  
CC = century

The value of date1 is not checked for a valid date with format1. However, CC always maintains the value specified in accordance with the THRESHOLD parameter. If you want date validation, use the DATEVAL routine before using DATECALC. The only valid Julian format is YYDDD.

The following are some, but not all, of the valid formats:

MMDDYY  
MMDDCCYY  
YYMMDD  
YYDDD (Julian)

{PLUS }  
{MINUS}

Specify whether the value of the days parameter is added to (PLUS) or subtracted from (MINUS) date1.

**Note:** DATECALC performs an arithmetic calculation. If you specify the PLUS keyword, and the value of the days is negative, the value is subtracted. Conversely, if you specify MINUS, and days is negative, the value is added.

days

Specify a numeric literal or a field that contains the value to be added or subtracted.

date2

Specify the name of the field to which the resulting date is written. The date is written using the format specified by the format2 parameter. A valid name is any previously defined field.

format2

Specify the format for date2.

[THRESHOLD value]

The THRESHOLD parameter is used to determine the century value if it is not supplied in the century format (CC) in the date. Specify a value that establishes the upper end of a one-hundred-year range in the 20th and 21st centuries used to control the CC portion of generated dates.

General rules for specifying THRESHOLD values are:

- The THRESHOLD value is ignored if you provide a century value (CC).
- If the dates to be generated do not exceed the year 2000, specify the THRESHOLD default value of 0. This causes all dates to have a range of 1901 through 2000.
- If the dates exceed the year 2000, choose a THRESHOLD high enough to generate correct dates in the 21st century, but not so high as to convert dates from the 20th century to the 21st century.
- When dates to be generated do not involve calculations for century, specify the THRESHOLD default value of 0.
- Valid values for THRESHOLD are 0 through 99.

For example, if THRESHOLD is 40, the upper boundary of the range is set to 2040, and the lower boundary is 1941. When converting YY to CCYY, each year is assigned a two-position century based on the range established by THRESHOLD. In this example, if year is 52, century is 19; if year is 21, century is 20.

It is important that the THRESHOLD value be correct for the range of dates to be generated. For example, if DATECALC is invoked to calculate dates between the years 1949 and 1952, and THRESHOLD is 50, the years 1949 and 1950 become 2049 and 2050, while the years 1951 and 1952 remain 1951 and 1952. In this respect, the YY (year) portion of the date controls the CC (century) portion in accordance with the THRESHOLD value.

## Operation - Inline

DATECALC generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

No change in the specification of parameters is required to use DATECALC with database files.

## Example

The following is an example of DATECALC.

Input

```
FILE ...
  INVOICE-DATE  1  6  N
  DUE-DATE      W  5  N
  ...
JOB ...
%DATECALC INVOICE-DATE MMDDYY PLUS 30 DUE-DATE YYDD
...
```

Results

The date contained in the field INVOICE-DATE is increased by 30 days, and the resulting date is converted from Gregorian format to Julian format and is written to the field DUE-DATE.

## DATECONV

The DATECONV routine converts a date in one format to any other date format. For example, you can convert month-day-year to year-month-day, Julian to Gregorian, and similar date conversions.

**Note:** Using non-numeric data or a zero for date fields results in a PAP299 error message which displays the field in error, along with its contents. Execution of the program stops, and a return code 32 is generated.

## Syntax

```
%DATECONV date1 format1 date2 format2 [THRESHOLD value]
```

date1

Specify the name of the field containing the date to be converted. The date in this field must be in the format specified by format1. The name of any previously defined numeric field is valid.

format1

Specify the format of the date1 field. Format1 is a literal description of pairs of letters. The letters indicate positions as follows:

```
MM = month
DD = day
YY = year
CC = century
```

The value of date1 is not checked for a valid date with the specified format. However, CC always maintains the value specified in accordance with the THRESHOLD parameter. If you want date validation, use the DATEVAL routine before using DATECONV. The only valid Julian format is YYDDD.

The following are some, but not all, of the valid formats:

MMDYY  
MMDCCYY  
YYMMDD  
YYDDD (Julian)

**Note:** For non-Julian dates, format1 must include the values MM, DD, and YY (in any order). The only valid Julian format is YYDDD.

date2

Specify the name of the field to which the converted date will be written. The date is written in the format specified by format2. A valid name is any previously defined field.

format2

Specify the format for the date2 field.

[THRESHOLD value]

The THRESHOLD parameter is used to determine the century value if it is not supplied in the century format (CC) in the date. Specify a value that establishes the upper end of a one-hundred-year range in the 20th and 21st centuries used to control the CC portion of generated dates.

General rules for specifying THRESHOLD values are:

- The THRESHOLD value is ignored if you provide a century value (CC).
- If the dates to be generated do not exceed the year 2000, specify the THRESHOLD default value of 0. This causes all dates to have a range of 1901 through 2000.
- If the dates exceed the year 2000, choose a THRESHOLD high enough to generate correct dates in the 21st century, but not so high as to convert dates from the 20th century to the 21st century.
- When dates to be generated do not involve calculations for century, specify the THRESHOLD default value of 0.
- Valid values for THRESHOLD are 0 through 99.

For example, if THRESHOLD is 40, the upper boundary of the range is set to 2040, and the lower boundary is 1941. When converting YY to CCYY, each year is assigned a two-position century based on the range established by THRESHOLD. In this example, if year is 52, century is 19; if year is 21, century is 20.

It is important that the THRESHOLD value be correct for the range of dates to be generated. For example, if DATECONV is invoked to convert dates between the years 1949 and 1952, and THRESHOLD is 50, the years 1949 and 1950 become 2049 and 2050, while the years 1951 and 1952 remain 1951 and 1952. In this respect, the YY (year) portion of the date controls the CC (century) portion in accordance with the THRESHOLD value.

## Operation - Inline

DATECONV generates no output, and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

No change in the specification of parameters is required to use DATECONV with database files.

## Example

The following is an example of DATECONV.

Input

```
FILE ...
  JULIAN-DATE   1  5  N
  GREG-DATE     W  6  N
  ...
JOB ...
%DATECONV JULIAN-DATE YYDDD GREG-DATE YYMDD
...
```

Results

The Julian date in the field named JULIAN-DATE is converted to Gregorian format, and the result is stored in the field named GREG-DATE.

## DATEGEN

The DATEGEN routine generates a date in a given format and writes it to the field parameter.

**Note:** The FILEGEN routine must be invoked to use the DATEGEN routine.

### Syntax

```
%DATEGEN field format [threshold] {BETWEEN date1 date2      }  
                                     {SEQUENCE date1 date2 increment}
```

field

Specify the name of the field where DATEGEN places the date it generates. The format of the generated date is specified by the format parameter. Valid names include any field defined in the file name specified in FILEGEN.

format

Specify the format of the date field. Format is a literal description of pairs of letters. The letters indicate positions as follows:

```
MM = month  
DD = date  
YY = year  
CC = century
```

Values for MM are limited to the numbers 01 through 12, values for DD are limited to 01 through 31, values for YY are limited to 00 through 99.

If values outside these limits are specified, the numbers will range only between the limits listed previously. YY must be specified whenever CC is specified. CC always maintains the value specified in accordance with the THRESHOLD parameter. You can specify MM, DD, YY, and CC in any order.

The following are some, but not all, of the valid formats:

```
MMDYY  
MMDCCYY  
YYMMDD  
YYDDD (Julian)
```

[threshold]

The THRESHOLD parameter is used to determine the century value if it is not supplied in the century format (CC) in the date. Specify a value that establishes the upper end of a one-hundred-year range in the 20th and 21st centuries used to control the CC portion of generated dates.

---

General rules for specifying THRESHOLD values are:

- The THRESHOLD value is ignored if you provide a century value (CC).
- If the dates to be generated do not exceed the year 2000, specify the THRESHOLD default value of 0. This causes all dates to have a range of 1901 through 2000.
- If the dates exceed the year 2000, choose a THRESHOLD high enough to generate correct dates in the 21st century, but not so high as to convert dates from the 20th century to the 21st century.
- When dates to be generated do not involve calculations for century, specify the THRESHOLD default value of 0.
- Valid values for THRESHOLD are 0 through 99.

For example, if THRESHOLD is 40, the upper boundary of the range is set to 2040, and the lower boundary is 1941. When converting YY to CCYY, each year is assigned a two-position century based on the range established by THRESHOLD. In this example, if year is 52, century is 19; if year is 21, century is 20.

It is important that the THRESHOLD value be correct for the range of dates to be generated. For example, if DATEGEN is invoked to generate dates between the years 1949 and 1952, and THRESHOLD is 50, the years 1949 and 1950 become 2049 and 2050, while the years 1951 and 1952 remain 1951 and 1952. In this respect, the YY (year) portion of the date controls the CC (century) portion in accordance with the THRESHOLD value.

{BETWEEN date1 date2}

The BETWEEN keyword causes a random date to be generated in the date span you specify by the date1 and date2 parameters. Date1 and date2 must be in the format specified by the format parameter. Dates generated are written to the field specified in the field parameter. Valid values for date1 and date2 are actual numeric values or the name of a field containing the value. The seed for the random generation of dates is obtained from FILEGEN.

Values generated are greater than or equal to the minimum and less than the maximum. If the desired range of dates is 010184 (MMDDYY) through 061684, specify a minimum of 010184 and a maximum of 061684.

```
{SEQUENCE date1 date2 increment}
```

The SEQUENCE keyword causes a fixed set of dates to be generated. The set of dates begins with date1 and is incremented for each record by the number of days specified by increment, until date2 is equaled or exceeded. The sequence is then repeated beginning with date1 until a date has been generated for each record being created.

Date1 and date2 must be in the format specified by the format parameter. Valid values for date1, date2, and increment are actual numeric values or the name of a field containing the value. To generate a decreasing set of dates, specify a date1 value greater than the date2 value and code a negative increment.

## Operation - Inline

Use DATEGEN only after you have specified the FILEGEN routine. DATEGEN can be specified with ALPHAGEN, NUMGEN, and BADGEN. Conditional execution of DATEGEN is discussed in [Example 3 - Conditional Execution of Data Generation Routines](#) following the BADGEN routine.

## Operation - Database

DATEGEN, with FILEGEN, cannot be used in a database application.

## Examples

The following example illustrates how DATEGEN is used with FILEGEN:

```
FILE CENTFILE F(24)
  NAME      1 12 A
  EMP#     13  5 N
  BIRTH    18  6 N
*
JOB INPUT NULL
%FILEGEN CENTFILE 25 5 NOHEX
%ALPHAGEN NAME ' CUST ' CONSTANT ' JOHNSON,SMITH,PETERS '
%NUMGEN EMP# SEQUENCE 1 10050 1
%DATEGEN BIRTH MMDDYY 0 BETWEEN 010151 010175
```

The following examples demonstrate two uses of DATEGEN. An example of the full facilities of the test data generation routines is found following the BADGEN routine.

```
BETWEEN
```

This example generates a random date between 010120 and 122960 for each record being created and writes it to the BIRTH field.

```
%DATEGEN BIRTH MMDDYY 0 BETWEEN 010120 123060
```

## SEQUENCE

A date of 12011999 is written to the EMPLOYED field of the first record, 12211999 to the second record, 01102000 to the third record, 01302000 to the fourth record, and so on. Each date is incremented by 20 days until the date 12012001 is equaled or exceeded. The sequence is then repeated beginning with the date 12011999.

```
%DATEGEN EMPLOYED MMDCCYY 50 SEQUENCE 12011999 12012001 20
```

## DATEVAL

The DATEVAL routine examines the content of a specified date field for a valid date in accordance with a specified date format. If the date field contains a valid date, the field DATEVAL-FLAG is set to the value YES. If the date field is invalid, the DATEVAL-FLAG is set to the value NO.

### Syntax

```
%DATEVAL field format [THRESHOLD value]
```

field

Specify the name of the field that contains the date being validated. Valid names include any previously defined numeric field.

format

The format for the comparison is a literal description of pairs of letters. The letters indicate positions as follows:

```
MM = month
DD = day
YY = year
CC = century
```

You can specify the letter pairs in any order. YY must be specified whenever you specify CC. The only valid Julian format is YYDDD.

The following are some, but not all, of the valid formats:

```
MMDYY
MMDCCYY
YYMMDD
YYDDD (Julian)
```

[THRESHOLD value]

The THRESHOLD parameter is used to determine the century value if it is not supplied in the century format (CC) in the date. Specify a value that establishes the upper end of a one-hundred-year range in the 20th and 21st centuries used to control the CC portion of generated dates.

General rules for specifying THRESHOLD values are:

- The THRESHOLD value is ignored if you provide a century value (CC).
- If the dates to be generated do not exceed the year 2000, specify the THRESHOLD default value of 0. This causes all dates to have a range of 1901 through 2000.
- If the dates exceed the year 2000, choose a THRESHOLD high enough to generate correct dates in the 21st century, but not so high as to convert dates from the 20th century to the 21st century.
- When dates to be generated do not involve calculations for century, specify the THRESHOLD default value of 0.
- Valid values for THRESHOLD are 0 through 99.

For example, if THRESHOLD is 40, the upper boundary of the range is set to 2040, and the lower boundary is 1941. When converting YY to CCYY, each year is assigned a two-position century based on the range established by THRESHOLD. In this example, if year is 52, century is 19; if year is 21, century is 20.

It is important that the THRESHOLD value be correct for the range of dates to be generated. For example, if DATEVAL is invoked to validate dates between the years 1949 and 1952, and THRESHOLD is 50, the years 1949 and 1950 become 2049 and 2050, while the years 1951 and 1952 remain 1951 and 1952. In this respect, the YY (year) portion of the date controls the CC (century) portion in accordance with the THRESHOLD value.

## Operation - Inline

The date field is compared to the specified format. For the comparison to be valid, the respective MM, DD, YY, and CC fields must contain valid values. For example, if the date field contains 043184, and the format field contains MMDDYY, the comparison is invalid because the DD (day) portion of the date exceeds 30 for the month of April. If the date field contains 022979, the comparison is invalid because 1979 is not a leap year.

DATEVAL does not produce a report. If the date field contains a valid date, an internal field DATEVAL-FLAG is set to the value YES. If the date field is invalid, the DATEVAL-FLAG is set to the value NO.

To perform further processing activities, you must code CA-Easytrieve Plus logic following the invocation of DATEVAL. You can code IF statements that test the DATEVAL-FLAG field.

For example, you may want to print a report of invalid dates, write all records with valid dates to an output file and perform further processing of the invalid dates, or any combination of events. Consult the CA-Easytrieve Plus *Reference Guide* for coding techniques. The example demonstrates coding using IF, DISPLAY, and END-IF statements.

## Operation - Database

No change in the specification of parameters is required to use DATEVAL with database files.

## Example

The following is an example of DATEVAL.

Input

```
FILE ...
  DATE          1 6 N
  INVOICE-NUM   7 4 P
  ...
JOB ...
%DATEVAL DATE MMDDYY
IF DATEVAL-FLAG EQ 'NO'
  DISPLAY +5 INVOICE-NUM +5 DATE
END-IF
...
```

Output

This example prints the invoice number and date for every record with an invalid date according to the format MMDDYY.

```
2983    083781
3953    023072
4263    063184
5337    131278
7654    000000
```

## DAYSAGO

**If you are using CA-Easytrieve Plus 6.2 or above, use DAYSAGOL.**

The DAYSAGO routine calculates the number of days that have elapsed between the current date and the date in a specified field. The result is compared to a value with a relational operator. If the calculated number of days satisfies the specified condition, the internal flag DAYSAGO-FLAG is set to the value YES. If the condition is not satisfied, DAYSAGO-FLAG is set to the value NO.

**Note:** DAYSAGOL functionality replaces DAYSAGO and supports a wider range of dates accurately. We recommend that systems running CA-Easytrieve Plus 6.2 or above use DAYSAGOL.

### Syntax

```
%DAYSAGO datefield format operator value [THRESHOLD value]
```

datefield

Specify the name of the field containing the date used for calculating the number of elapsed days. The date in this field must be in the format specified by the format parameter. A valid name is any previously defined field.

format

Specify the format of datefield. Format is a literal description of pairs of letters. The letters indicate positions as follows:

```
MM = month  
DD = day  
YY = year  
CC = century
```

The value of datefield is not checked for a valid date with the specified format. If you want date validation, use the DATEVAL routine before using DAYSAGO. The only valid Julian format is YYDDD.

The following are some, but not all, of the valid formats:

```
MMDDYY  
MMDDCCYY  
YYMMDD  
YYDDD (Julian)
```

operator

Specify any relational operator (EQ, NE, LT, LE, GT, or GE).

value

The number of days between the date in datefield and the current date is compared to this numerical value. For value, you can specify an actual numeric value or the name of a field containing the numeric value.

[THRESHOLD value]

The THRESHOLD parameter is used to determine the century value if it is not supplied in the century format (CC) in the date. Specify a value that establishes the upper end of a one-hundred-year range in the 20th and 21st centuries used to control the CC portion of generated dates.

General rules for specifying THRESHOLD values are:

- The THRESHOLD value is ignored if you provide a century value (CC).
- If the dates to be generated do not exceed the year 2000, specify the THRESHOLD default value of 0. This causes all dates to have a range of 1901 through 2000.
- If the dates exceed the year 2000, choose a THRESHOLD high enough to generate correct dates in the 21st century, but not so high as to convert dates from the 20th century to the 21st century.
- When dates to be generated do not involve calculations for century, specify the THRESHOLD default value of 0.
- Valid values for THRESHOLD are 0 through 99.

For example, if THRESHOLD is 40, the upper boundary of the range is set to 2040, and the lower boundary is 1941. When converting YY to CCYY, each year is assigned a two-position century based on the range established by THRESHOLD. In this example, if year is 52, century is 19; if year is 21, century is 20.

It is important that the THRESHOLD value be correct for the range of dates to be generated. For example, if DAYSAGO is invoked to process dates between the years 1949 and 1952, and THRESHOLD is 50, the years 1949 and 1950 become 2049 and 2050, while the years 1951 and 1952 remain 1951 and 1952. In this respect, the YY (year) portion of the date controls the CC (century) portion in accordance with the THRESHOLD value.

## Operation - Inline

DAYSAGO does not produce a report. If the calculated number of days satisfies the condition specified, the internal flag DAYSAGO-FLAG is set to the value YES. If the condition is not satisfied, DAYSAGO-FLAG is set to the value NO.

To perform further processing activities, you must code CA-Easytrieve Plus logic following the invocation of DAYSAGO. Code IF and END-IF statements, which test the DAYSAGO-FLAG field around the logic you want to perform. The example demonstrates this coding technique.

## Operation - Database

No change in the specification of parameters is required to use DAYSAGO with database files.

## Example

The following is an example of DAYSAGO.

Input

```
FILE ...
  EMPLOYEE-NAME   1 20  A
  SSN              21 11  A
  HIREDATE         32  6  N
  ...
JOB ...
%DAYSAGO HIREDATE MMDDYY LE 120
IF DAYSAGO-FLAG EQ 'YES'
  PRINT DAYSAGO-REPORT
END-IF
...
REPORT DAYSAGO-REPORT
TITLE 1 'EMPLOYEES HIRED SINCE JANUARY 1984'
LINE EMPLOYEE-NAME SSN HIREDATE
```

Output

```
      EMPLOYEES HIRED SINCE JANUARY 1984
EMPLOYEE-NAME      SSN      HIREDATE
PETER   BRENNON    324-23-5467  022484
CHARLES   JENSEN    342-23-0232  031284
BETTY    WALTON     384-64-8547  010384
JENNIFER WILSON     311-42-7472  033084
```

This example examines a file to list all employees hired in the last 120 days. The DAYSAGO-FLAG is examined, and a report listing the employee name, social security number, and date hired is printed if DAYSAGO-FLAG contains the value YES.

## DAYSGOL

The DAYSGOL routine calculates the number of days that have elapsed between the current date and the date in a specified field. The result is compared to a value with a relational operator. If the calculated number of days satisfies the specified condition, the internal flag DAYSAGO-FLAG is set to the value YES. If the condition is not satisfied, DAYSAGO-FLAG is set to the value NO.

**Note:** DAYSGOL functionality replaces DAYSAGO and supports a wider range of dates accurately. We recommend that you use DAYSGOL in place of DAYSAGO in systems running CA-Easytrieve Plus 6.2 or above.

### Syntax

```
%DAYSGOL datefield format operator value [THRESHOLD value]
```

datefield

Specify the name of the field containing the date used for calculating the number of elapsed days. The date in this field must be in the format specified by the format parameter. A valid name is any previously defined field.

format

Specify the format of datefield. Format is a literal description of pairs of letters. The letters indicate positions as follows:

```
MM = month
DD = day
YY = year
CC = century
```

The value of datefield is not checked for a valid date with the specified format. If you want date validation, use the DATEVAL routine before using DAYSGOL. The only valid Julian format is YYDDD.

The following are some, but not all, of the valid formats:

```
MMDYY
MMDCCYY
YYMMDD
YYDDD (Julian)
```

operator

Specify any relational operator (EQ, NE, LT, LE, GT, or GE).

value

The number of days between the date in datefield and the current date is compared to this numerical value. For value, you can specify an actual numeric value or the name of a field containing the numeric value.

[THRESHOLD value]

The THRESHOLD parameter is used to determine the century value if it is not supplied in the century format (CC) in the date. Specify a value that establishes the upper end of a one-hundred-year range in the 20th and 21st centuries used to control the CC portion of generated dates.

General rules for specifying THRESHOLD values are:

- The THRESHOLD value is ignored if you provide a century value (CC).
- If the dates to be generated do not exceed the year 2000, specify the THRESHOLD default value of 0. This causes all dates to have a range of 1901 through 2000.
- If the dates exceed the year 2000, choose a THRESHOLD high enough to generate correct dates in the 21st century, but not so high as to convert dates from the 20th century to the 21st century.
- When dates to be generated do not involve calculations for century, specify the THRESHOLD default value of 0.
- Valid values for THRESHOLD are 0 through 99.

For example, if THRESHOLD is 40, the upper boundary of the range is set to 2040, and the lower boundary is 1941. When converting YY to CCYY, each year is assigned a two-position century based on the range established by THRESHOLD. In this example, if year is 52, century is 19; if year is 21, century is 20.

It is important that the THRESHOLD value be correct for the range of dates to be generated. For example, if DAYSAGOL is invoked to process dates between the years 1949 and 1952, and THRESHOLD is 50, the years 1949 and 1950 become 2049 and 2050, while the years 1951 and 1952 remain 1951 and 1952. In this respect, the YY (year) portion of the date controls the CC (century) portion in accordance with the THRESHOLD value.

## Operation - Inline

DAYSAGOL does not produce a report. If the calculated number of days satisfies the condition specified, the internal flag DAYSAGO-FLAG is set to the value YES. If the condition is not satisfied, DAYSAGO-FLAG is set to the value NO.

To perform further processing activities, you must code CA-Easytrieve Plus logic following the invocation of DAYSAGOL. Code IF and END-IF statements, which test the DAYSAGO-FLAG field around the logic you want to perform. The example demonstrates this coding technique.

## Operation - Database

No change in the specification of parameters is required to use DAYSAAGOL with database files.

## Example

The following is an example of DAYSAAGOL.

Input

```
FILE ...
  EMPLOYEE-NAME   1 20  A
    SSN           21 11  A
    HIREDATE      32  6  N
  ...
JOB ...
%DAYSAAGOL HIREDATE MMDDYY LE 120
IF DAYSAGO-FLAG EQ 'YES'
  PRINT DAYSAAGOL-REPORT
END-IF
...
REPORT DAYSAAGOL-REPORT
TITLE 1 'EMPLOYEES HIRED SINCE JANUARY 1984'
LINE EMPLOYEE-NAME SSN HIREDATE
```

Output

```
      EMPLOYEES HIRED SINCE JANUARY 1984
      EMPLOYEE-NAME          SSN          HIREDATE
      PETER   BRENNON   324-23-5467   022484
      CHARLES  JENSEN   342-23-0232   031284
      BETTY    WALTON   384-64-8547   010384
      JENNIFER WILSON   311-42-7472   033084
```

This example examines a file to list all employees hired in the last 120 days. The DAYSAGO-FLAG is examined, and a report listing the employee name, social security number, and date hired is printed if DAYSAGO-FLAG contains the value YES.

## DAYSCALC

The DAYSCALC routine calculates the number of elapsed days between two specified dates of any format. The calculation is:

days = date1 - date2

### Syntax

%DAYSCALC date1 format1 date2 format2 days [THRESHOLD value]

date1

Specify the name of the field containing the date that date2 is subtracted from. The date in this field must be in the format specified by format1. A valid name is any previously defined field.

format1

Specify the format of the date1 field. Format1 is a literal description of pairs of letters. The letters indicate positions as follows:

MM = month  
DD = day  
YY = year  
CC = century

The value of date1 is not checked for a valid date with the specified format. If you want date validation, use the DATEVAL routine before using DAYSCALC. The only valid Julian format is YYDDD.

The following are some, but not all, of the valid formats:

MMDDYY  
MMDDCCYY  
YYMMDD  
YYDDD (Julian)

date2

Specify the name of the field containing the date to be subtracted. The date in this field must be in the format specified by format2. A valid field name is any previously defined field.

format2

Specify the format for the date2 field.

days

Specify the name of the numeric field that will hold the results of the calculation. The value of days is negative if date1 is earlier than date2. However, a negative indicator for days is printed only if it is defined with decimal positions (0 through 18) or with a user-defined edit mask containing a negative number indicator. For additional information on edit masks, see the *CA-Easytrieve Plus Reference Guide*.

You must define the days field before invoking DAYSCALC.

[THRESHOLD value]

The THRESHOLD parameter is used to determine the century value if it is not supplied in the century format (CC) in the date. Specify a value that establishes the upper end of a one-hundred-year range in the 20th and 21st centuries used to control the CC portion of generated dates.

General rules for specifying THRESHOLD values are:

- The THRESHOLD value is ignored if you provide a century value (CC).
- If the dates to be generated do not exceed the year 2000, specify the THRESHOLD default value of 0. This causes all dates to have a range of 1901 through 2000.
- If the dates exceed the year 2000, choose a THRESHOLD high enough to generate correct dates in the 21st century, but not so high as to convert dates from the 20th century to the 21st century.
- When dates to be generated do not involve calculations for century, specify the THRESHOLD default value of 0.
- Valid values for THRESHOLD are 0 through 99.

For example, if THRESHOLD is 40, the upper boundary of the range is set to 2040, and the lower boundary is 1941. When converting YY to CCYY, each year is assigned a two-position century based on the range established by THRESHOLD. In this example, if year is 52, century is 19; if year is 21, century is 20.

It is important that the THRESHOLD value be correct for the range of dates to be generated. For example, if DAYSCALC is invoked to process dates between the years 1949 and 1952, and THRESHOLD is 50, the years 1949 and 1950 become 2049 and 2050, while the years 1951 and 1952 remain 1951 and 1952. In this respect, the YY (year) portion of the date controls the CC (century) portion in accordance with the THRESHOLD value.

## Operation - Inline

DAYSCALC generates no output. It can be used alone, with other routines, with CA-Easytrieve Plus logic, or with both.

Use DAYSCALC only on dates during the years 1800 through 2199.

## Operation - Database

No change in the specification of parameters is required to use DAYSCALC with database files.

## Example

The following is an example of DAYSCALC.

Input

```

FILE ...
  END-DATE      1  6  N
  START-DATE    7  6  N
  ...
RESULT          W 10 P 0
JOB ...
%DAYSCALC END-DATE MMDDYY START-DATE MMDDYY RESULT
  ...
  
```

Results

The number of elapsed days between the fields END-DATE and START-DATE is calculated, and the result is stored in a field called RESULT. RESULT is defined with 0 decimal places to ensure that a negative sign will print if the field is negative.

---

## DIVIDE

The DIVIDE routine calculates the integer quotient and remainder of two numbers. For example,  $6/4 = 1$  remainder 2. This routine is not intended to replace the CA-Easytrieve Plus division operation ( $\setminus$ ), but is intended primarily to produce a remainder in those cases in which it is of significance.

### Syntax

```
%DIVIDE  number  divisor  quotient  remainder
```

number

Specify the number to be divided. It can be an actual numeric value or a previously defined field containing a numeric value.

divisor

Specify the value by which the number is divided. This must be a nonzero integer. It can be a numeric value or a previously defined field containing a numeric value.

quotient

Specify the field to which the quotient of the calculation is written. A valid name is any previously defined numeric field.

remainder

Specify the field to which the remainder of the calculation is written. A valid name is any previously defined numeric field.

### Operation - Inline

DIVIDE is designed for use with integer fields only. Results of the calculation are incorrect if decimal places are defined.

DIVIDE generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

### Operation - Database

No change in specification of parameters is required to use DIVIDE with database files.

## Example

The following is an example of DIVIDE. In this example, quotients and remainders were calculated successfully until the field DI contained a zero.

Input

```
FILE INFILE
  NU      1 2 N
  DI      4 2 N
*
DEFINE QU  W 4 N
DEFINE RE  W 4 N
*
JOB INPUT INFILE
*
%DIVIDE NU DI QU RE
  DISPLAY NU ' / ' DI ' = ' QU ' REMAINDER ' RE
```

Output

```
02 / 04 = 0000 REMAINDER 0002
12 / 03 = 0004 REMAINDER 0000
07 / 02 = 0003 REMAINDER 0001
09 / 01 = 0009 REMAINDER 0000
09 / 02 = 0004 REMAINDER 0001
09 / 03 = 0003 REMAINDER 0000
09 / 04 = 0002 REMAINDER 0001
09 / 05 = 0001 REMAINDER 0004
09 / 06 = 0001 REMAINDER 0003
09 / 07 = 0001 REMAINDER 0002
09 / 08 = 0001 REMAINDER 0001
09 / 09 = 0001 REMAINDER 0000
09 / 89 = 0000 REMAINDER 0009
09 / 92 = 0000 REMAINDER 0009
73 / 09 = 0008 REMAINDER 0001
00 / 12 = 0000 REMAINDER 0000
07 / 40 = 0000 REMAINDER 0007
01 / 04 = 0000 REMAINDER 0001
17 / 13 = 0001 REMAINDER 0004
06 / 04 = 0001 REMAINDER 0002
***** - PAP308 DIVISION BY ZERO  DI
```

## DOLUNIT

The DOLUNIT routine performs a dollar unit sampling of the input file, DOLUNIT selects records for sampling according to monetary units rather than physical attributes. It optionally creates a sample file based on the selections made during dollar unit processing. You can print a report including the input parameters, the makeup of the sample file, and the positive, negative, and total book values of the file.

## Syntax

```
%DOLUNIT1 infile field width cutoff seed [VALUE {ABS}] [REPORT {YES}]
[ {ACT}] [ {NO }]
[ {POS}] [ { }]

%DOLUNIT2 {outfile} [TOP {topfile}] [KEY {keyfile}] [DBFILE infile] +
{NOFILE } [ {NOFILE } ] [ {NOFILE } ]
[PERFORM procname]
```

infile

Specify the name of the input file to DOLUNIT. A valid name is any previously defined file.

field

Specify the name of the quantitative field from which the values for the dollar unit sampling are taken. A valid name is any quantitative field defined in the input file.

width

Specify the value of the cell width for the sample. This determines the target value that must be exceeded for a record to be selected for the sample file. A valid value is an actual numeric value or the name of a field containing a numeric value. Values can contain up to two decimal places. Values greater than two decimal places are truncated on the right. (Cell width is analogous to the target value parameter of the SPS routine.)

cutoff

Specify the value for cutoff for the top stratum. Any record having a value greater than or equal to the cutoff becomes part of the top stratum and is sent to the sample file. To direct the top stratum samples to a separate file, use the TOP parameter. A valid value is an actual numeric value or the name of a field containing a numeric value. Values can contain up to two decimal places. Values with greater than two decimal places are truncated on the right.

seed

Specify an arbitrary number that initiates the random number generator. This seed is used to randomize the updated target value for each cell. A valid value is an actual numeric value or the name of a field containing a numeric value. Values can be up to seven digits in length with no decimal places. Values greater than seven digits are truncated on the left.

```
[VALUE {ABS }]  
[      {ACT }]  
[      {POS }]
```

This optional parameter controls the value for the input field used in the sampling process. The default value is ABS.

**ABS**—Specifies that the absolute value of the input field is used in the sampling process. This method places equal emphasis on both positive and negative values.

**ACT**—Specifies that the actual value of the input field is used in the sampling process. This method places emphasis on the net value of positive and negative values.

**POS**—Specifies that only the values of the input field that are greater than zero are used in the sampling process. This method places emphasis on the positive values.

```
[REPORT {YES }]  
[      {NO  }]
```

This optional parameter specifies whether the DOLUNIT report is produced. Specify NO to inhibit the printing of the report. The default is YES, which produces the report.

```
{outfile }  
{NOFILE }
```

Specify whether records selected for the sample are to be written to an output file.

**outfile**—Records selected for the sample are written to the file indicated by outfile. File characteristics must be coded on the FILE statement for this output file. Outfile must have the same file characteristics as the input file, or outfile must have the appropriate file characteristics to be able to accommodate the longest input record. Valid names for outfile include any previously defined file.

**NOFILE**—Records selected for the sample are not written to an output file.

```
[TOP {topfile }]  
[   {NOFILE }]
```

This optional parameter specifies the name of the file to which the top stratum records (audited values that exceed the cutoff) are written. File characteristics must be coded on the FILE statement for this output file.

**topfile**—Topfile must have the same file characteristics as the input file, or topfile must have the appropriate file characteristics to be able to accommodate the longest input record. Valid names for topfile include any previously defined file. The default is for top stratum records to be written to the file indicated by the outfile parameter. For use of this parameter, see [Operation - Stand-alone DISPLAY](#) in this routine.

**NOFILE**—This option is valid only for database use of DOLUNIT. NOFILE specifies that top stratum records are to be written to the file indicated by the outfile parameter.

---

```
[KEY {keyfile }]  
[  {NOFILE  }]
```

Key records are those records known to be significant (such as error-prone records or records with an unusual history or specific values). This optional parameter specifies the name of the file to which key records are to be written. Use of this parameter prevents key records from appearing on the sample file. File characteristics must be coded on the FILE statement for this output file.

**keyfile**–Keyfile must have the same file characteristics as the input file, or keyfile must have the appropriate file characteristics to be able to accommodate the longest input record. Valid names for keyfile include any previously defined file. For use of this parameter, see [Operation - Stand-alone DISPLAY](#) in this routine.

**NOFILE**–This option is valid only for database use of DOLUNIT. NOFILE specifies that no key processing occurs.

```
[DBFILE infile]
```

Specify this optional parameter only for database use of DOLUNIT. Specify the name of the input file to DOLUNIT. The name must be the same name that you specified for infile on the first invocation statement.

```
[PERFORM procname]
```

Specify the name of a CA-Easytrieve Plus procedure that is performed by the DOLUNIT routine after each record is selected or not selected for an output file. If a record is selected for an output file, the internal field DOLUNIT-SELECTED is set to the value YES. If a record is not selected for an output file, DOLUNIT-SELECTED is set to the value NO. The internal field name is DOLUNIT-SELECTED for the output file that the outfile/NOFILE parameter identifies, DOLUTOP-SELECTED for the output file that topfile identifies, and DOLUKEY-SELECTED for the output file that keyfile identifies.

After the invocation of DOLUNIT2, you can define a CA-Easytrieve Plus procedure to perform processing based on whether the input record is selected for an output file.

For example, the procedure can test the DOLUTOP-SELECTED field and display appropriate fields of the input record if the value is YES. This provides a listing of all records selected for the topfile in addition to the normal report that DOLUNIT produces. For a description of the format and use of a procedure, see the CA-Easytrieve Plus *Reference Guide*. For an example of the use of this parameter, see the chapter “[Advanced Techniques](#).”

If you specify this parameter, it must follow any occurrence of the TOP or KEY parameters. This is an optional parameter. If you do not specify the name, the system substitutes a default procname which is a dummy procedure that performs no processing.

## Operation - Stand-alone DISPLAY

DOLUNIT can be used without the KEY or TOP options. If a separate file for top stratum items is required, use the TOP parameter with the associated topfile file name. If key items are to be separated into a different file, use the KEY parameter with the associated keyfile parameter.

If both TOP and KEY parameters are specified, the one that is specified first on the macro invocation statement takes precedence. For example, if a record satisfies the conditions for both top stratum and key processing, it is written to the sample file indicated by the parameters TOP or KEY (whichever is specified first).

Key records are identified by using an IF statement. If the KEY parameter is specified, IF, PERFORM DOLUKEY, and END-IF statements must be placed between the %DOLUNIT1 and %DOLUNIT2 statements.

The following example writes all records to the file KEYFILE that have the field EMPLOYEE-CODE equal to 99:

```
FILE INFILE ...
  PAY ...
  EMPLOYEE-CODE ...
...
FILE KEYFILE ...
...
FILE OUTFILE ...
...
%DOLUNIT1 INFILE PAY 10000 2000 1357
IF EMPLOYEE-CODE EQ 99
  PERFORM DOLUKEY
END-IF
%DOLUNIT2 OUTFILE KEY KEYFILE
```

Two sample files are created. Records from the dollar unit algorithm that are sampled are written to the file OUTFILE. Records with EMPLOYEE-CODE equal to 99 are not used in the dollar unit algorithm and are written to KEYFILE.

The following example is the same as the previous example except that screening logic is added:

```
FILE INFILE ...
  PAY ...
  EMPLOYEE-CODE ...
...
FILE KEYFILE ...
...
FILE OUTFILE ...
...
%DOLUNIT1 INFILE PAY 10000 2000 1357
IF EMPLOYEE-CODE EQ 0
  GO TO JOB
END-IF
IF EMPLOYEE-CODE EQ 99
  PERFORM DOLUKEY
END-IF
%DOLUNIT2 OUTFILE KEY KEYFILE
```

The IF, GO TO JOB, and END-IF statements comprise the screening logic. These statements bypass any records with EMPLOYEE-CODE equal to zero. The IF, PERFORM DOLUKEY, and END-IF statements comprise the key processing logic. It does not matter whether the screening logic is coded before or after the key logic. A record is skipped if it satisfies the conditions for the screening logic. Each set of logic must be coded with the IF statement, followed by the GO TO JOB (screening) or PERFORM DOLUKEY (key processing), and the END-IF statement.

## Operation - Database

The DBFILE parameter identifies DOLUNIT as a routine that can access database files. This is an optional parameter that you need not specify for nondatabase use. However, you must specify this parameter when using DOLUNIT in a database application. Furthermore, you must specify all parameters on the second invocation statement, with the exception of TOP and KEY, in the order shown in the description of the syntax. This restriction on parameter placement applies only to the database use of DOLUNIT.

When you specify DBFILE, you must specify both TOP and KEY. If you do not want a TOP file, specify NOFILE. This causes all records for the top stratum to be written to the file that the outfile parameter identifies. If you do not want a KEY file, specify NOFILE and do not code a PERFORM DOLUKEY statement. This prevents records from being written to a KEY file.

The requirement that you specify the TOP and KEY parameters applies only to the database use of DOLUNIT.

## Example

The following is an example of DOLUNIT.

Input

```
FILE PAYFILE ...
  GROSS ...
  DEPT ...
  ...
FILE OUTFL ...
  ...
FILE TOPFL ...
  ...
FILE KEYFL ...
  ...
%DOLUNIT1 PAYFILE GROSS 1000000 60000 135
IF DEPT EQ 21
  PERFORM DOLUKEY
END-IF
%DOLUNIT2 OUTFL TOP TOPFL KEY KEYFL
```

## Output

## DOLLAR UNIT SAMPLING REPORT

## INPUT PARAMETERS

INPUT FILENAME	PAYFILE
INPUT FIELD	GROSS
VALUE OF INPUT FIELD IS	ABS
CELL WIDTH	1,000,000.00
TOP STRATUM CUTOFF	60,000.00

## SAMPLE FILE(S)

NUMBER OF CELLS PROCESSED	37
---------------------------	----

	RECORD COUNT	FILE NAME	TOTAL VALUE
NOT SELECTED	1,157		34,860,801.00
GENERAL SAMPLE	36	OUTFL	1,319,107.00
TOP STRATUM	8	TOPFL	481,743.00
KEY VALUE	13	KEYFL	392,251.00
TOTAL SAMPLES	57		2,193,101.00
PROCESSED TOTAL	1,214		37,053,902.00

## ABSOLUTE VALUE TOTAL

	37,046,280.00	POSITIVE BOOK VALUE
MINUS	7,622.00-	NEGATIVE BOOK VALUE
	-----	
	37,053,902.00	TOTAL BOOK VALUE - ABSOLUTE

## ACTUAL VALUE TOTAL

	37,046,280.00	POSITIVE BOOK VALUE
PLUS	7,622.00-	NEGATIVE BOOK VALUE
	-----	
	37,038,658.00	TOTAL BOOKVALUE - ACTUAL

The DOLUNIT report:

- Lists the input parameters, the results of the dollar unit sampling, and positive, negative, and total book values of the file.
- Identifies the number of records in all appropriate areas, including the total number of samples.
- Provides the count for key processing, depending on whether the KEY parameter is specified.
- Lists the file names of the general, top, and key sample files.
- Calculates a processed total of all records that participated in the dollar unit sampling. This total is the actual value of those records that participated.

The positive and negative book values are the actual values accumulated by the routine. The negative book values are then subtracted and added to yield the absolute and actual value totals. The value used in the dollar unit sampling algorithm is based upon the VALUE parameter: absolute, actual or positive.

## DUPTTEST

The DUPTTEST routine tests a field to determine if the content is identical in more than one record. It prints a report of all duplicate records. Two options for output are available:

- A detailed report of each duplicate record, with one field to identify the record and one field to provide additional information.
- A summary report, listing only the total number of duplicate records detected.

With either option, the duplicate records can be written to an output file.

### Syntax

```
%DUPTTEST1 infile [LRECL length]
%DUPTTEST2 infile {S} field {outfile} {field2 field3}
                  {U}      {NOFILE } {SUMMARY      }
```

infile

Specify the name of the input file to DUPTTEST. A valid name is any previously defined file.

[LRECL length]

Optionally specify the length of the input record. The default is 32,767 bytes. If the record length is less than 32,767, you can improve the efficiency of both disk storage utilization and execution speed by specifying the exact length of the record using the following example:

```
infile-lrecl + 1 work byte + 4 RDW bytes = LRECL
```

```
{S}
{U}
```

Specify whether the records input to DUPTTEST are sorted or unsorted.

**S**—Indicates that records are sorted in order by the field parameter. The sorted order can be in ascending or descending sequence.

**U**— Indicates that records are not in sorted order. DUPTEST will sequence these records in temporary storage in ascending order by the field parameter before it begins the comparison process.

field

Specify the name of the field for which duplicates are searched. If the content of this field is the same in two or more records, a duplicate condition exists, and the activities specified in subsequent parameters are performed. A valid name is any nonquantitative field defined in the input file.

```
{outfile }
{NOFILE }
```

Specify whether an output file of duplicate records is created.

**outfile**— Duplicate records are written to the output file indicated by this parameter. File characteristics must be coded on the FILE statement for this output file. Valid names for outfile include any previously defined file.

**NOFILE**— Duplicate records are not written to an output file.

```
{field2 field3 }
{SUMMARY }
```

Specify whether the report will contain a detail line for each duplicate record or a summary report giving the total of duplicates.

**field2 field3**— When this option is specified, three fields are listed for each duplicate record:

- Field (the field for which duplicates are searched)
- Field2
- Field3

Valid names are any previously defined fields.

**SUMMARY**— When this option is specified, a summary report is produced. This consists of the total number of sets of duplicates and the total of all duplicate records in the file.

## Operation - Stand-alone REPORT

Use the DUPTEST routine to test for duplicate records in a file. A detail or summary report is produced, depending on the option you specify. If specified, duplicate records can be written to an output file.

## Operation - Database

DUPTEST can access both database and nondatabase files without any changes in the specification of parameters. The infile parameter can be a nondatabase file name or the name of a database file defined in the library section.

## Example

The following is an example of DUPTEST.

Input

```
FILE CUSTFIL ...
  INVNO      1      5      N
  NAME       7     15      A
  BALANCE   23      7      N 2
...
%DUPTTEST1 CUSTFIL
%DUPTTEST2 CUSTFIL S INVNO NOFILE NAME BALANCE
```

Output

The report lists three sets of records with duplicate invoice numbers. In each case, the total of any numeric field is listed in addition to the tally for each set of duplicates.

4/20/88	REPORT OF DUPLICATE RECORDS	PAGE 1		
	KEY IS INVNO			
	LISTING NAME AND BALANCE			
	NOFILE IS PRODUCED			
	DUPLICATE	DUPLICATE	DUPLICATE	NUMBER OF
	INVNO	NAME	BALANCE	DUPLICATES
	10134	LARRY JONES	123.64	
		MARY JAFFEY	436.34	
	INVNO TOTAL		559.98	2
	63674	CRAIG HALL	563.67	
		CRAIG HILL	563.67	
	INVNO TOTAL		1,127.34	2
	98723	GARY CONDREN	1,504.66	
		KATHY BRADY	564.60	
		REX THOMPSON	44.33	
	INVNO TOTAL		2,113.59	3
	FINAL TOTAL		3,800.91	7



# Generalized/Statistical Routines

## E-W

This chapter lists alphabetically, and gives detailed descriptions of, routines EACHNTH through WEEKDAY.

### EACHNTH

The EACHNTH routine selects a sample from an existing file, based on a specified starting point and subsequent selection of every *n*th record. Generally, this method does not produce a statistically valid sample because each record does not have an equal probable selection. However, it is useful in certain types of compliance testing.

#### Syntax

```
%EACHNTH1  infile interval [START num]
%EACHNTH2  {outfile} [DBFILE infile] [PERFORM procname]
           {NOFILE }
```

*infile*

Specify the name of the input file to EACHNTH. A valid name is any previously defined file.

*interval*

Specify the interval between records selected from the file. For example, if 5 is specified, every 5th record will be selected. A valid value is an actual numeric value or the name of a field containing the numeric value.

[START num]

Use this optional parameter to start selecting records at a record other than 1. A valid value is an actual numeric value or the name of a field containing the numeric value.

```
{outfile }  
{NOFILE }
```

Specify whether an output file of selected records is created.

**outfile**—Selected records are written to the output file indicated by this parameter. File characteristics must be coded on the FILE statement for this output file. Valid names for outfile include any previously defined file.

**NOFILE**—Selected records are not written to an output file.

```
[DBFILE infile]
```

You must specify this option parameter only for database use of EACHNTH. Specify the name of the input file to EACHNTH. The name must be the same name that was specified for infile on the first invocation statement.

```
[PERFORM procname]
```

Specify the name of a CA-Easytrieve Plus procedure which is performed by the EACHNTH routine after each record is selected or not selected for the sample file. The internal field, EACHNTH-SELECTED, is set to YES if a record is selected for the sample file and to NO if a record is not selected.

After the invocation of EACHNTH2, you can define a CA-Easytrieve Plus procedure to perform processing based on whether the input record was selected for the sample file. For example, the procedure could test the EACHNTH-SELECTED field and display the appropriate fields of the input record if the value is YES. This provides a listing of selected records in addition to the normal report that EACHNTH produces. For a description of the format and use of a procedure, see the CA-Easytrieve Plus *Reference Guide*. For an example of the use of this parameter, see the chapter "[Advanced Techniques](#)."

This is an optional parameter. If you do not specify it, the system substitutes a default procname which is a dummy procedure that performs no processing.

## Operation - Stand-alone DISPLAY

EACHNTH provides a simple way to create a subset of the records in a file. However, because EACHNTH does not produce a statistically valid sample, it is important that you know the number of records contained in the input file. You set the interval at which records are selected. EACHNTH will select records until the end of the input file is reached.

## Operation - Database

The DBFILE parameter identifies EACHNTH as a routine that can access database files. This is an optional parameter that you need not specify for nondatabase use. However, you must specify this parameter when using EACHNTH in a database application. Furthermore, you must specify all parameters on the second invocation statement, in the order shown in the description of the syntax. This restriction applies only to the database use of EACHNTH.

## Example

The following is an example of EACHNTH.

Input

```
FILE PAYFILE FB (44 4400)
*
FILE OUTFILE FB (44 4400)
*
%EACHNTH1 PAYFILE 5 START 4
%EACHNTH2 OUTFILE
```

Output

```
                EACHNTH SAMPLING REPORT
                INPUT PARAMETERS
INPUT FILENAME          PAYFILE
INTERVAL SIZE           5
START RECORD            4

                SAMPLE FILE
NUMBER OF RECORDS PROCESSED      1000
NUMBER OF RECORDS SELECTED       10

FILE OUTFILE WILL BE CREATED
```

## EXPO

The EXPO routine calculates the result of raising a number to a specified power.

## Syntax

```
%EXPO value exponent result [VALDEC dec1] [RESDEC dec2]
```

value

Specify the numeric value of the field being raised to the specified power. A valid value is an actual numeric value or the name of a field containing a numeric value. Values with decimal places more than defined by VALDEC are truncated to the right.

exponent

Specify the power to which the value parameter is raised. A valid value is an actual numeric value or the name of a field containing a numeric value. Values can contain up to four decimal places and are truncated to the right if more than four are specified.

result

Specify the field to which the result of the calculation is placed. A valid name is any previously defined field. Fields with more decimal places defined in RESDEC are not accurate past the number of decimal places in RESDEC.

[VALDEC dec1]

Specify the number of decimal places needed in value. The default is 2. The maximum number of decimal places is 15.

[RESDEC dec2]

Specify the number of decimal places needed in result. The default is 4. The maximum number of decimal places is 15.

## Operation - Inline

EXPO generates no output, and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

An error condition can occur when the first parameter (value) is too large or the result is too large for the field definition. An error condition also occurs when the first parameter (value) is negative, and the second parameter (exponent) is not a whole number. When this occurs, the exponent is truncated to an integer, and the routine returns the value of YES in an internal field called EXPO-ERROR. The EXPO routine monitors these errors. The routine returns the value of YES in EXPO-ERROR for either error.

If you suspect that problems in the specification of the parameters may have occurred, check the field EXPO-ERROR after invoking the EXPO routine.

## Operation - Database

No change in the specification of parameters is required to use EXPO with database files.

## Example

The following is an example of EXPO.

Input

```
FILE ...
  NUMBER      1  6  N  2
  EXPONENT    7  6  N  4
  RESULT      W 10  N  4
...
JOB ...
%EXPO NUMBER EXPONENT RESULT
IF EXPO-ERROR EQ 'YES'
  DISPLAY 'ERROR DETECTED IN FOLLOWING RESULTS:'
END-IF
PRINT EXPO-REPORT
REPORT EXPO-REPORT
TITLE 1 'EXPONENTIATION LISTING'
LINE NUMBER EXPONENT RESULT
```

Output

In this example, the results of raising the field NUMBER to the power specified by EXPONENT are calculated. The field EXPO-ERROR is examined for the described error condition, and an appropriate message is printed in the report.

```
EXPONENTIATION LISTING

NUMBER      EXPONENT      RESULT
20.10       3.2000      14,798.8301
14.60       1.8000      124.6910
3.40-       3.0000      39.3040-
ERROR DETECTED IN FOLLOWING RESULTS:
3.40-       3.2000      39.3040-
.           .           .
.           .           .
.           .           .
4.80       5.2000      3,487.0200
```

## FILECOMP

The FILECOMP routine compares specified locations in two files and produces a report of mismatched records. An optional key matching facility can be used to realign the files after a mismatch occurs.

### Syntax

```
%FILECOMP primary secondary maximum { ALL  
                                         { 'loc1,len1,loc2,len2,...,locx,lenx' } } +  
    [ HEX      ] [PRIKEYS 'pkey1 pkey2 ... pkey6'] +  
    [ NOHEX    ]  
    [          ]  
    [SECKEYS 'skey1 skey2 ... skey6']
```

#### primary

Specify the name of the primary input file. If record-matching keys are used, you must define the key fields specified in PRIKEYS in this file. A valid name is any previously defined file.

#### secondary

Specify the name of the secondary input file. If record-matching keys are used, you must define the key fields specified in SECKEYS in this file. A valid name is any previously defined file.

#### maximum

Specify the total number of record pairs that can be unequal before the file compare terminates. A valid value for maximum is an actual numeric value or the name of a field containing a numeric value.

```
{ ALL  
{ 'loc1,len1,loc2,len2,...,locx,lenx' }
```

This parameter describes the locations in the records in the primary and secondary file that will participate in the file comparison.

**ALL**—Specifies that the full length of each record is to be compared, beginning with position 1.

**loc1,len1,loc2,len2,...,locx,lenx**—Specifies that certain portions of each record are to be compared.

The areas are defined using couplets:

- The first number designates the relative location in the record.
- The second number designates the length for which the compare is to be performed.

The entire string of numbers is enclosed in single quotes, and entries are separated by commas. No spaces are allowed.

**Note:** You do not have to specify the couplets in ascending order according to location. For example, it may be more efficient to compare locations 74 in each record pair first, and then location 6. The compare of a record pair terminates, and records are selected for printing as soon as a mismatch in any position being compared is detected.

[HEX ]  
[NOHEX]

This optional parameter specifies whether you want a hexadecimal printout of the mismatched records included in the report.

**HEX**— A hexadecimal printout of mismatched records is included in the report. HEX is the default value.

**NOHEX**— The hexadecimal printing of records is eliminated from the report.

[PRIKEYS 'pkey1 pkey2 ... pkey6']

This optional parameter specifies the key fields used to realign the files when a mismatch occurs. You must define the fields pkey1 pkey2 ... pkey6 in the primary file. The parameters must be separated by a blank, and the entire string of key fields must be enclosed in single quotation marks.

The number of key fields specified in PRIKEYS must equal the number of key fields specified in SECKEYS. Six is the maximum number of key fields. A valid name is any alphanumeric or numeric field defined in the primary input file.

[SECKEYS 'skey1 skey2 ... skey6']

This optional parameter specifies the key fields used to realign the files when a mismatch occurs. You must define the fields skey1 skey2 ... skey6 in the secondary file. The parameters must be separated by a blank, and the entire string of key fields must be enclosed in single quotation marks.

The number of key fields specified in SECKEYS must equal the number of key fields specified in PRIKEYS. The maximum number of key fields is six. A valid name is any alphanumeric or numeric field defined in the secondary input file.

## Operation - Stand-alone DISPLAY

Key fields defined in PRIKEYS and SECKEYS must be defined in the primary and secondary files. If the number of key fields defined in PRIKEYS and SECKEYS is not equal, an error message is printed, and execution stops. If PRIKEYS and SECKEYS are not specified, no file realignment is attempted. This means that if a mismatch occurs because of an unequal record pair, the remainder of the file may be interpreted as being mismatched, unless records become realigned by chance.

## Operation - Database

FILECOMP cannot be used in a database application.

## Examples

The following are two examples of the FILECOMP routine.

### Example One

This example demonstrates a file comparison of all fields in the input files. No keys are used, so the files are not automatically realigned if they become offset by the addition or deletion of a record from either file.

Input

```
FILE INPUT1 ...
  field-name ...
...
FILE INPUT2 ...
  field-name ...
...
%FILECOMP INPUT1 INPUT2 5 ALL
```

Output

```
***** PAPER220 - *UNEQUAL PAIR*                1  IN POSITION
17
   RECORD NUMBER          4  FROM INPUT1
CHAR CUSTOMER000000004    121083    00016
ZONE CEEEDDCDFEEEEFF00000200FFFFF00000270FFFFF
NUMR 342364590000000040000130C1210830000270C00016
     1...5...10...15...20...25...30...35...40...4

   RECORD NUMBER          4  FROM INPUT2
CHAR CUSTOMER000000007    121083    00016
ZONE CEEEDDCDFEEEEFF00000200FFFFF00000270FFFFF
NUMR 342364590000000070000130C1210830000270C00016
     1...5...10...15...20...25...30...35...40...4
```

```

**** PAP220 - *UNEQUAL PAIR*                2 IN POSITION
17
   RECORD NUMBER          8 FROM INPUT1
CHAR CUSTOMER000000008 - 040184 00024
ZONE CEEEDDCDFEEEE00000060FFFFFF00000490FFFF
NUMR 34236459000000080000810C0401840000590C00024
   1...5...10...15...20...25...30...35...40...4
   RECORD NUMBER          8 FROM INPUT2
CHAR CUSTOMER000000003 - 040184 00024
ZONE CEEEDDCDFEEEE00000060FFFFFF00000490FFFF
NUMR 34236459000000030000810C0401840000590C00024
   1...5...10...15...20...25...30...35...40...4
**** PAP220 - *UNEQUAL PAIR*                3 IN POSITION
16
   RECORD NUMBER          11 FROM INPUT1
CHAR CUSTOMER000000011 / 111883 00030
ZONE CEEEDDCDFEEEE00000680FFFFFF00000670FFFF
NUMR 342364590000000110000010C1118830000420C00030
   1...5...10...15...20...25...30...35...40...4

   RECORD NUMBER          11 FROM INPUT2
CHAR CUSTOMER000000021 / 111883 00030
ZONE CEEEDDCDFEEEE00000680FFFFFF00000670FFFF
NUMR 342364590000000210000010C1118830000420C00030
   1...5...10...15...20...25...30...35...40...4

**** PAP212 - END OF FILE REACHED ON INPUT1
**** PAP213 - END OF FILE REACHED ON INPUT2
**** PAP216 - FILE COMPARE ENDED
                TOTAL OF          14 INPUT1 RECORDS READ
                TOTAL OF          14 INPUT2 RECORDS READ
**** PAP218 - TOTAL OF          3 UNEQUAL RECORDS FOUND

```

For each nonmatching pair of records, an informational message lists the accumulated number of nonmatching records, the position of the mismatching information, the record number and file name for the primary and secondary files, and a hexadecimal listing of each record. After all records are listed, other messages indicate when the end of file was encountered for both files, whether the compare ended normally or the count of records exceeded the maximum parameter, the total number of records in each file, and the total number of unequal records.

## Example Two

This example demonstrates a file comparison of specified fields with the use of keys for realigning the files after an unequal pair of records is found. This example is similar to Example One, except that the files are realigned after a mismatch occurs. A message is printed indicating that end-of-file was first reached on INPUT1. The remainder of the records are unmatched.

Input

```

FILE INPUT1 ...
  IN1-DEPNO    13  5  N
  IN1-NAME     1 12  A
  ...
FILE INPUT2

```

```

IN2-DEPNO      13  5  N
IN2-NAME       1 12  A
...
%FILECOMP INPUT1 INPUT2 10 '13,5,40,5' PRIKEYS 'IN1-DEPNO IN1-NAME' -
                SECKEYS 'IN2-DEPNO IN2-NAME'

```

Output

```

***** PAP220 - *UNEQUAL PAIR*                1 IN KEY A
      RECORD NUMBER          5 FROM INPUT1
CHAR CUSTOMER000000004      020484      00018
ZONE CEEEDDCDFEEEE00000170FFFFFF00000790FFFF
NUMR 34236459000000040000700C0204840000600C00018
      1..5...10...15...20...25...30...35...40...4

      RECORD NUMBER          5 FROM INPUT2
CHAR CUSTOMER000000005      020484      00018
ZONE CEEEDDCDFEEEE00000170FFFFFF00000790FFFF
NUMR 34236459000000050000700C0204840000600C00018
      1..5...10...15...20...25...30...35...40...4

***** PAP220 - *UNEQUAL PAIR*                2 IN KEY A
      RECORD NUMBER          6 FROM INPUT1
CHAR CUSTOMER000000008      040584      R 00020
ZONE CEEEDDCDFEEEE0000000FFFFFF00000970FFFF
NUMR 34236459000000080000310C0405840000990C00020
      1..5...10...15...20...25...30...35...40...4

      RECORD NUMBER          5 FROM INPUT2
CHAR CUSTOMER000000005      020484      00018
ZONE CEEEDDCDFEEEE00000170FFFFFF00000790FFFF
NUMR 34236459000000050000700C0204840000600C00018
      1..5...10...15...20...25...30...35...40...4

***** PAP212 - END OF FILE REACHED ON INPUT1
***** PAP214 - ALL REMAINING RECORDS ARE UNMATCHED
CHAR CUSTOMER000000014      & 030784      00032
ZONE CEEEDDCDFEEEE00000150FFFFFF00000700FFFF
NUMR 342364590000000140000980C0307840000770C00032
      1..5...10...15...20...25...30...35...40...4

CHAR CUSTOMER000000015      022084      - 00034
ZONE CEEEDDCDFEEEE00000430FFFFFF00000760FFFF
NUMR 342364590000000150000490C0220840000790C00034
      1..5...10...15...20...25...30...35...40...4

***** PAP216 - FILE COMPARE ENDED
                        TOTAL OF          11 INPUT1 RECORDS READ
                        TOTAL OF          13 INPUT2 RECORDS READ
***** PAP218 - TOTAL OF          4 UNEQUAL RECORDS FOUND

```

---

## FILEGEN

The FILEGEN routine specifies the output file to which data, created by data generation routines, is written.

### Syntax

```
%FILEGEN file number seed {HEX }  
                                {NOHEX}
```

file

Specify the name of the output file to which generated data is to be written. It corresponds to the name on the FILE statement in the library section of your program. A valid name is any previously defined file.

number

Specify the number of output records for which data is generated. Execution stops when the specified number of records have been generated. Valid values include an actual numeric value greater than 0 and less than 100,000,000.

seed

Specify an arbitrary number that initiates the internal random number generator for each data generation routine. If a different seed is specified in a rerun of an otherwise identical job, different data is generated. If the same seed is specified in a rerun of a job, the same data is generated. A valid value is an actual numeric value or the name of a field containing a numeric value. Values can be up to seven digits in length with no decimal places. Values greater than seven digits will be truncated on the left.

```
{HEX }  
{NOHEX}
```

Specify whether a hexadecimal listing of generated data is produced.

**HEX**— A hexadecimal listing of generated data is produced.

**NOHEX**— No listing is produced.

### Operation - Inline

FILEGEN is invoked only once and must precede the data generation routines ALPHAGEN, DATEGEN, NUMGEN, or BADGEN. There is no input file to FILEGEN, so screening of input data is not allowed.

## Operation - Database

FILEGEN and the associated routines ALPHAGEN, DATEGEN, NUMGEN, and BADGEN cannot be used in a database application.

## Example

The following example illustrates the use of FILEGEN. An example of its use with the data generation routines is given following the discussion of the BADGEN routine.

Input

```
FILE PAYFILE ...
  field-name ...
JOB INPUT NULL
...
%FILEGEN PAYFILE 500 17823 NOHEX
...
```

This example establishes the file name as PAYFILE and specifies that 500 records will be created. The seed for use in data generation routines is 17823, and a hexadecimal listing of the output file is not printed.

If the HEX option is specified, the output from FILEGEN can consist of a hexadecimal listing of the records created with the data generation routines.

## FLDVALR

The FLDVALR routine validates a field against a specified range of values. A report of valid or invalid records is produced. Optionally, an output file of valid or invalid records may also be produced. If the field contains valid data, the field FLDVAL-SELECTED is set to VALID. If the field contains invalid data, the field FLDVAL-SELECTED is set to INVALID.

## Syntax

```
%FLDVALR1 infile
%FLDVALR2 infile field { VALID } low high +
                    { INVALID }

                    RPTSELECT { VALID } { field2 field3 } +
                    { INVALID } { SUMMARY }

[ FILE outfile FILESELECT {VALID } ]
[                               {INVALID } ]
```

infile

Specify the name of the input file to FLDVALR. A valid name is any previously defined file.

field

Specify the name of the field to be validated. A valid name is any field defined in the input file.

```
{ VALID }
{ INVALID }
```

Specify VALID if the range is to be considered a valid range. Specify INVALID if the range is to be considered an invalid range.

low

Low is the lower limit of the range of values. Low can be a literal value or the name of a field containing the value. If the field being validated is alphanumeric, the literal must be placed in triple quotes ("literal"). The comparison between field and value is made under the rules of the CA-Easytrieve Plus Field Relational Condition. See the CA-Easytrieve Plus *Reference Guide* for complete rules.

high

High is the upper limit of the range of values. High can be a literal value or the name of a field containing the value. If the field being validated is alphanumeric, the literal must be placed in triple quotes ("literal"). The comparison between field and value is made under the rules of the CA-Easytrieve Plus Field Relational Condition. See the CA-Easytrieve Plus *Reference Guide* for complete rules.

```
RPTSELECT { VALID } {field2 field3 }
          { INVALID} {SUMMARY      }
```

RPTSELECT defines the contents of the validation report.

**VALID**—When VALID is specified, the report will contain only valid records.

**INVALID**—When INVALID is specified, the report will contain only invalid records.

**field2 field3**—Specify field2 and field3 to request a detail report. This option lists three fields for each record in the file:

- Field (the field being validated)
- Field2
- Field3

Valid names are any previously defined field.

**SUMMARY**—Specify SUMMARY to request a report consisting of the total number of records selected for the input file.

```
[FILE outfile FILESELECT {VALID   }]  
[                          {INVALID }]
```

This optional parameter specifies if an output file is to be created. Selected records are written to the output file indicated by this parameter. File characteristics must be coded on the FILE statement for this output file. A valid name for outfile is any previously defined file. FILESELECT specifies if the output file is to contain valid or invalid records.

**VALID**—When VALID is specified, the output file will contain only valid records.

**INVALID**—When INVALID is specified, the output file will contain only invalid records.

## Operation - Stand-alone REPORT

FLDVALR reads the input file and determines whether the designated field contains valid or invalid databases upon the defined parameters.

FLDVALR will automatically produce a validation report as defined by the RPTSELECT parameters.

Optionally, an output file of valid or invalid records can also be produced by coding the FILE parameters.

## Operation - Database

FLDVALR can access database and nondatabase files without any changes in the specification of parameters. The infile parameter can be a nondatabase file name or the name of a database file defined in the library section.

## Example

The following is an example of FLDVALR.

Input

```
FILE PERSNL FB(150 1800)
  BRANCH  1  1 N
  REGION  2  2 N
  NAME    17 20 A
DEFINE LOW-VALUE W 1 A VALUE 'A'
DEFINE HIGH-VALUE W 1 A VALUE 'F'
%FLDVALR1 PERSNL
%FLDVALR2 PERSNL NAME VALID LOW-VALUE HIGH-VALUE RPTSELECT VALID +
  REGION BRANCH
```

Output

```
FIELD VALIDATION REPORT
INPUT FILE: PERSNL      FIELD COMPARED: NAME      SELECT: VALID RANGE
REPORT OF VALID RECORDS  NO OUTPUT FILE IS PRODUCED
```

	NAME	REGION	BRANCH
	BERG NANCY	02	1
	CORNING GEORGE	03	1
	ARNOLD LINDA	04	1
	BRANDOW LYDIA	01	1
	BYER JULIE	04	1
	DENNING RALPH	03	2
	EPERT LINDA	03	3
	CROCI JUDY	04	3

## FLDVALT

The FLDVALT routine validates a field against a specified table of values. A report of valid or invalid records is produced. Optionally, an output file of valid or invalid records may also be produced. If the field contains valid data, the field FLDVAL-SELECTED is set to VALID. If the field contains invalid data, the field FLDVAL-SELECTED is set to INVALID.

## Syntax

```
%FLDVALT1 infile
%FLDVALT2 infile field {VALID } table desclength +
  {INVALID}

RPTSELECT {VALID } {field2 field3} +
  {INVALID} {SUMMARY }

[FILE outfile FILESELECT {VALID }
[                               {INVALID}]
```

infile

Specify the name of the input file to FLDVALT. A valid name is any previously defined file.

field

Specify the name of the field to be validated. A valid name is any field defined in the input file.

{VALID }  
{INVALID}

Specify VALID if the table contains valid values. Specify INVALID if the table contains invalid values.

table

Specify the name of the table to be used for validation. A valid name is any previously defined table.

desclength

Specify the length of the table's description field (used to define the length of FLDVALT-DESC). If a field is found to be valid, the table description is available in the field FLDVALT-DESC.

RPTSELECT {VALID } {field2 field3}  
{INVALID} {SUMMARY }

RPTSELECT defines the contents of the validation report.

**VALID**—When VALID is specified, the report will contain only valid records.

**INVALID**—When INVALID is specified, the report will contain only invalid records.

**field2 field3**—Specify field2 and field3 to request a detail report. This option lists three fields for each record in the file:

- Field (the field being validated)
- Field2
- Field3

Valid names are any previously defined field.

**SUMMARY**—Specify SUMMARY to request a report consisting of the total number of records selected for the input file.

```
[FILE outfile FILESELECT {VALID  }]
[                               {INVALID}]
```

This optional parameter specifies if an output file is to be created. Selected records are written to the output file indicated by this parameter. File characteristics must be coded on the FILE statement for this output file. A valid name for outfile is any previously defined file. FILESELECT specifies if the output file is to contain valid or invalid records.

**VALID**—When VALID is specified, the output file will contain only valid records.

**INVALID**—When INVALID is specified, the output file will contain only invalid records.

## Operation - Stand-alone REPORT

FLDVALT reads the input file and determines whether the designated field contains valid or invalid databased upon a table lookup.

The FLDVALT routine searches for a match to any entry in the table specified in the table parameter. This table must be defined in a FILE statement, using the TABLE keyword. The table is either instream or external (a data file). For example:

```
FILE VALTTBL TABLE INSTREAM
ARG 1 2 N
DESC 3 8 A
01REGION 1
02REGION 2
03REGION 3
ENDTABLE
```

See the *CA-Easytrieve Plus Reference Guide* for complete information on table processing.

If a field is found to be valid (FLDVAL-SELECTED is VALID), the table description (for example, REGION 1) is moved to the field FLDVALT-DESC.

FLDVALT will automatically produce a validation report as defined by the RPTSELECT parameters.

Optionally, an output file of valid or invalid records can also be produced by coding the FILE parameters.

## Operation - Database

FLDVALT can access both database and nondatabase files without any changes in the specification of parameters. The infile parameter can either be a nondatabase file name or the name of a database file defined in the library section.

## Example

The following is an example of FLDVALT.

Input

```
FILE INFILE F(80)
  DATE 1 6 N HEADING ('GREGORIAN' 'DATE') MASK 'Z9/99/99'
  MONTH 1 2 N
  DAY 3 2 N
  YEAR 5 2 N
FILE VALTBL TABLE INSTREAM
  ARG 1 2 N
  DESC 3 10 A
01 JANUARY
02 FEBRUARY
03 MARCH
10 OCTOBER
11 NOVEMBER
12 DECEMBER
ENDTABLE
*
%FLDVALT1 INFILE
%FLDVALT2 INFILE MONTH VALID VALTBL 10 RPTSELECT VALID +
  FLDVALT-DESC DATE
```

Output

```
FIELD VALIDATION REPORT
INPUT FILE: INFILE      FIELD COMPARED: MONTH      SELECT: VALID TABLE
REPORT OF VALID RECORDS  NO OUTPUT FILE IS PRODUCED
```

MONTH	FLDVALT-DESC	GREGORIAN DATE
01	JANUARY	1/01/89
02	FEBRUARY	2/02/89
03	MARCH	3/03/89
01	JANUARY	1/02/89
12	DECEMBER	12/31/89
11	NOVEMBER	11/23/89

## FLDVALV

The FLDVALV routine validates a field against a specified value.

A report of valid or invalid records is produced. Optionally, an output file of valid or invalid records may also be produced. If the field contains valid data, the field FLDVAL-SELECTED is set to VALID. If the field contains invalid data, the field FLDVAL-SELECTED is set to INVALID.

### Syntax

```
%FLDVALV1 infile
%FLDVALV2 infile field { VALID  } value      +
                    { INVALID }
                    RPTSELECT { VALID  } { field2 field3 } +
                    { INVALID } { SUMMARY  }
[ FILE outfile FILESELECT {VALID  } ]
[                               {INVALID } ]
```

infile

Specify the name of the input file to FLDVALV. A valid name is any previously defined file.

field

Specify the name of the field to be validated. A valid name is any field defined in the input file.

```
{VALID  }
{INVALID }
```

Specify VALID if the range is to be considered a valid range. Specify INVALID if the range is to be considered an invalid range.

value

Value can be a literal value or the name of a field containing the value. If the field being validated is alphanumeric, the literal must be placed in triple quotes ("literal"). The comparison between field and value is made under the rules of the CA-Easytrieve Plus Field Relational Condition. See the CA-Easytrieve Plus *Reference Guide* for complete rules.

```
RPTSELECT {VALID } { field2 field3 }  
          {INVALID} { SUMMARY }
```

RPTSELECT defines the contents of the validation report.

**VALID**—When VALID is specified, the report will contain only valid records.

**INVALID**—When INVALID is specified, the report will contain only invalid records.

**field2 field3**—Specify field2 and field3 to request a detail report. This option lists three fields for each record in the file:

- Field (the field being validated)
- Field2
- Field3

Valid names are any previously defined field.

**SUMMARY**—Specify SUMMARY to request a report consisting of the total number of records selected for the input file.

```
[FILE outfile FILESELECT {VALID }]  
[ {INVALID }]
```

This optional parameter specifies if an output file is to be created. Selected records are written to the output file indicated by this parameter. File characteristics must be coded on the FILE statement for this output file. A valid name for outfile is any previously defined file. FILESELECT specifies if the output file is to contain valid or invalid records.

**VALID**—When VALID is specified, the output file will contain only valid records.

**INVALID**—When INVALID is specified, the output file will contain only invalid records.

## Operation - Stand-alone REPORT

FLDVALV reads the input file and determines whether the designated field contains valid or invalid databases upon the defined parameters.

FLDVALV will automatically produce a validation report as defined by the RPTSELECT parameters.

Optionally, an output file of valid or invalid records can also be produced by coding the FILE parameters.

## Operation - Database

FLDVALV can access database and nondatabase files without any changes in the specification of parameters. The infile parameter can be a nondatabase file name or the name of a database file defined in the library section.

## Example

The following is an example of FLDVALV.

Input

```
FILE PERSNL FB(150 1800)
  BRANCH  1  1 N
  REGION  2  2 N
  NAME    17 20 A
DEFINE TEST-VALUE W  1 N VALUE 2
%FLDVALV1 PERSNL
%FLDVALV2 PERSNL REGION VALID TEST-VALUE RPTSELECT VALID +
  BRANCH NAME
```

Output

```
FIELD VALIDATION REPORT
INPUT FILE: PERSNL      FIELD COMPARED: REGION      SELECT: VALID VALUE
REPORT OF VALID RECORDS  NO OUTPUT FILE IS PRODUCED
```

REGION	BRANCH	NAME
02	1	BERG NANCY
02	1	NAGLE MARY
02	2	POWELL CAROL
02	2	KRUSE MAX
02	3	THOMPSON JANICE
02	3	SMOTH CINDY
02	3	ISAAC RUTH
02	3	LACH LORRIE
02	3	GRECO LESLIE
02	3	REYNOLDS WILLIAM
02	4	JOHNSON LISA
02	4	HAFER ARTHUR

## GAPCHCK

The GAPCHCK routine tests a numeric field to determine if any records in a continuous sequence are missing from the file. A report that lists any numbers missing from the sequence is automatically generated. For a series of missing numbers, the first and last numbers are printed. This listing provides information to verify missing items.

### Syntax

```
%GAPCHCK1      infile      field [LRECL length]
```

```
%GAPCHCK2      {S}  
                {U}      [DBFILE infile]  
                { }
```

`infile`

Specify the name of the input file to GAPCHCK. A valid name is any previously defined file.

`field`

Specify the name of the numeric field being tested.

`[LRECL length]`

Optionally specify the length of the input record. The default is 32,767 bytes. If the record length is less than 32,767, you can improve the efficiency of both disk storage utilization and execution speed by specifying the exact length of the record using the following formula:

$$\text{infile-lrecl} + 1 \text{ work byte} + 4 \text{ RDW bytes} = \text{LRECL}$$

`{S}`  
`{U}`

Specify whether the records input to GAPCHCK are sorted or unsorted.

**S**—Indicates that the records are sorted in order by the field parameter. The sorted order must be in ascending sequence.

**U**—Indicates that records are not in sorted order. GAPCHCK sequences these records in temporary storage in ascending order by the field parameter before it begins the comparison process.

`[DBFILE infile]`

This is an optional parameter. You must specify this parameter only for database use of GAPCHCK. Specify the name of the input file to test. The name must be the same name that you specified for `infile` on the first invocation statement.

## Operation - Stand-alone REPORT

GAPCHCK assumes the input file contains a continuous sequence of numbers in the designated field. You must specify whether the file is already in ascending sequence for the designated field. If necessary, GAPCHCK will sort the file. GAPCHCK then reads the file and determines whether the designated field contains any missing numbers.

GAPCHCK will automatically produce a report listing all missing numbers.

## Operation - Database

The DBFILE parameter identifies GAPCHCK as a routine that can access database files. This is an optional parameter that you need not specify for nondatabase use. However, you must specify this parameter when using GAPCHCK in a database application. Furthermore, you must specify all parameters on the second invocation statement, in the order shown in the description of the syntax. This restriction on parameter placement applies only to the database use of GAPCHCK.

## Example

This example uses GAPCHCK to sort an unsorted file and produce a report of missing numbers.

Input

```
FILE BUILD
CHECK#          1   4   N
DESCRIPT        5  60  A
*
%GAPCHCK1 BUILD CHECK#
%GAPCHCK2 U
```

Output

```
6/25/90          NUMERIC GAP TEST REPORT          PAGE    1
                  RANGE OF NUMBERS FOR    BUILD
                  0001   ---   0100
                  MISSING NUMBERS
                  0004
                  0010   ---   0011
                  0017   ---   0020
                  0032   ---   0034
                  0036
                  0038   ---   0040
                  0054   ---   0059
                  0061
                  0065
```

## GETDATE

**If you are using CA-Easytrieve Plus 6.2 or above, use GETDATEL.**

The GETDATE routine obtains the current date from the system and places it in the specified field. The resulting date contains no slashes, hyphens or other non-numeric characters.

### Syntax

```
%GETDATE field
```

field

Specify the name of the field to which the current date is placed. The date will be in the same format that is selected when the Toolkit system is installed. A valid field name is any previously defined field that will hold six characters.

### Operation - Inline

GETDATE generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

### Operation - Database

No change in the specification of parameters is required to use GETDATE with database files.

### Example

The following is an example of GETDATE.

Input

```
...  
DEFINE DATEFIELD  W  6  N  
JOB ...  
...  
%GETDATE DATEFIELD  
...
```

Results

GETDATE stores the current date in the field named DATEFIELD. The format of the value in DATEFIELD is defined by the option selected during Toolkit installation.

---

## GETDATEL

The GETDATEL routine obtains the current date, in the CA-Easytrieve Plus SYSDATE-LONG format, and places it in the specified field. The resulting date contains no slashes, hyphens, or other non-numeric characters.

**Note:** GETDATEL functionality replaces GETDATE and supports a wider range of dates accurately. We recommend that systems running CA-Easytrieve Plus 6.2 or above use GETDATEL instead of GETDATE. GETDATEL is supported only in CA-Easytrieve Plus 6.2 and above.

### Syntax

```
%GETDATEL field
```

field

Specify the name of the field to which the current date is placed. The date will be in the same format as SYSDATE-LONG. A valid field name is any previously defined field that will hold eight characters.

### Operation - Inline

GETDATEL generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

### Operation - Database

No change in the specification of parameters is required to use GETDATEL with database files.

### Example

The following is an example of GETDATEL.

Input

```
...  
DEFINE DATEFIELD   W   8   N  
JOB ...  
...  
%GETDATEL DATEFIELD  
...
```

## Results

GETDATEL stores the current date in the field named DATEFIELD. The format of the value in DATEFIELD is defined by the option selected during Toolkit installation.

## MULTDUP

The MULTDUP routine compares specified fields within records in the input file to determine if duplicates exist. You can compare from one to 50 fields, referred to as keys. Records are selected as duplicates when all of the specified key fields match. Duplicate records can optionally be written to an output file.

MULTDUP differs from DUPTEST in that it permits you to examine multiple fields for duplicates, instead of just one field.

### Syntax

```
%MULTDUP1 infile [LRECL length]
%MULTDUP2 infile {S} { outfile } key1 key2 ... keyn
                {U} { NOFILE  }
```

infile

Specify the name of the input file to MULTDUP. A valid name is any previously defined file.

[LRECL length]

Optionally specify the length of the input record. The default is 32,767 bytes. If the record length is less than 32,767, you can improve the efficiency of both disk storage utilization and execution speed by specifying the exact length of the record using the following formula:

$$\text{infile-lrecl} + 1 \text{ work byte} + 4 \text{ RDW bytes} = \text{LRECL}$$

{S}  
{U}

Specify whether the records input to MULTDUP are sorted or unsorted.

**S**—Indicates that records are sorted in the order specified by the key fields. The sorted order can either be in ascending or descending sequence. For more details about sort requirements, see [Operation - Stand-alone REPORT](#).

**U**— Indicates that records are not in sorted order. MULTDUP will sequence the records in temporary storage in the order specified by the key fields before it begins the comparison process.

```
{outfile}
{NOFILE }
```

Specify whether an output file of duplicate records is to be created.

**outfile**— Duplicate records are written to the output file indicated by this parameter. File characteristics must be coded on the FILE statement for this output file. Valid names for outfile include any previously defined file.

**NOFILE**— Duplicate records are not written to an output file.

```
key1 key2 ... keyn
```

List the fields that are to participate in the search for duplicate records. Records are selected as duplicates only when all of the specified key fields match. You can specify a maximum of 50 key fields. A valid key nonquantitative field is any field defined in the input file.

## Operation - Stand-alone REPORT

Before MULTDUP can test for duplicates, input records must be sorted according to the comparison fields. For example, if two key fields are specified, records must be sorted by key1 and by the key2 field in key1. Key1 is referred to as the major sort field, and key2 the minor sort field. If sort indicator S is specified, MULTDUP assumes that the file is sorted. If sort indicator U is specified, MULTDUP sorts the records according to the key1 through key $n$  parameters.

The MULTDUP routine does not produce a report. For each duplicate condition detected, an internal flag MULTDUP-FLAG is set to the value YES. If no duplicate condition exists, the MULTDUP-FLAG is set to the value NO.

To create a listing of duplicate records, code the appropriate IF and END-IF statements to test the MULTDUP-FLAG field immediately following the invocation of MULTDUP. When the flag is YES, the CA-Easytrieve Plus statements PRINT or DISPLAY can be used to create the desired listing (see the CA-Easytrieve Plus *Reference Guide*). Examples of report processing with MULTDUP can be found in the examples that follow.

Even though MULTDUP does not contain a REPORT statement, it functions as a stand-alone REPORT routine. This means that, even though a display statement is used for output (see [Example One](#)), MULTDUP still has all the limitations of a stand-alone REPORT routine.

## Operation - Database

MULTDUP can access both database and nondatabase files without any changes in the specification of parameters. The infile parameter can be a nondatabase file name or the name of a database file defined in the library section.

## Examples

The following two examples demonstrate the use of MULTDUP.

### Example One

Input

```
FILE ACCOUNT
FIRSTNM      1  8  A
MIDDLE       9  1  A
LAST         10 10  A
ACCOUNT-NUM  20  7  N
...
%MULTDUP1 ACCOUNT
%MULTDUP2 ACCOUNT U NOFILE LAST FIRSTNM MIDDLE
IF MULTDUP-FLAG EQ 'YES'
    DISPLAY +5, ACCOUNT-NUM, +5, FIRSTNM, +1, MIDDLE, +1, LAST
END-IF
...
```

Output

This example demonstrates the use of the DISPLAY statement to create a listing of duplicate records. When a duplicate record is detected, the MULTDUP-FLAG field is set to the value YES.

```
0283747      JOAN R      JONES
0343346      JOAN R      JONES
0745785      LARRY L      LITTLE
0748658      LARRY L      LITTLE
0798223      LARRY L      LITTLE
0103467      JOHN L WILLIAMSON
0187645      JOHN L WILLIAMSON
```

The IF statement tests for the YES condition and displays four fields if a duplicate record is found. The first field printed is an identification field, such as an account number, to assist in identifying the duplicate record. The remaining fields are the actual duplicate fields.

## Example Two

Input

```

FILE ACCOUNT
  FIRSTNM      1  8  A
  MIDDLE       9  1  A
  LAST        10 10  A
  ACCOUNT-NUM  20  7  N
FILE DUPFILE
...
%MULTDUP1 ACCOUNT
%MULTDUP2 ACCOUNT S DUPFILE LAST FIRSTNM MIDDLE
IF MULTDUP-FLAG EQ 'YES'
  PRINT DUPLICATE-REPORT
END-IF
REPORT DUPLICATE-REPORT
TITLE 1 'DUPLICATE RECORDS ON THE CUSTOMER FILE'
HEADING ACCOUNT-NUM 'ACCOUNT NUMBER'
HEADING FIRSTNM 'FIRST'
LINE ACCOUNT-NUM FIRSTNM MIDDLE LAST

```

Output

This example demonstrates the use of the PRINT statement to create a listing of duplicate records. When a duplicate record is located, the MULTDUP-FLAG field is set to the value YES.

The IF statement tests for the YES condition and executes the report named DUPLICATE-REPORT if a duplicate record is found. DUPLICATE-REPORT automatically prints page numbers, column headings, and the report title as specified. For information on report processing parameters, see the *CA-Easytrieve Plus Reference Guide*.

```

3/14/84      DUPLICATE RECORDS ON THE CUSTOMER FILE      PAGE  1
              ACCOUNT NUMBER  FIRST  MIDDLE  LAST
              0283747         JOAN   R        JONES
              0343346         JOAN   R        JONES
              0745785         LARRY  L        LITTLE
              0748658         LARRY  L        LITTLE
              0798223         LARRY  L        LITTLE
              0103467         JOHN   L        WILLIAMSON
              0187645         JOHN   L        WILLIAMSON

```

## NUMGEN

The NUMGEN routine generates numeric data and writes it to the field parameter.

**Note:** The FILEGEN routine must be invoked to use the NUMGEN routine.

### Syntax

```
%NUMGEN field      {BETWEEN   minimum maximum      }  
                   {SEQUENCE  from to increment  }  
                   {CONSTANT  'value1,value2,...,valuen' }
```

field

Specify the name of the field where NUMGEN places the numeric data it generates. Valid names include any numeric field, defined in the file name specified in FILEGEN, with a data format of N (zoned decimal), P (packed decimal), B (binary), or U (unsigned packed decimal). The maximum number of generated digits for field is 15.

{BETWEEN minimum maximum}

The BETWEEN keyword causes random numbers to be generated for the field you specified in the field parameter. Generated values will be in the range you specify by the minimum and maximum parameters. Valid values for minimum and maximum are actual numeric values or the name of a field containing the value. Minimum and maximum must be less than 100 trillion and can contain up to four decimal places. The seed for the random generation of values is obtained from the FILEGEN routine.

Values generated are greater than or equal to the minimum and less than the maximum. If the desired range of values is 10.75 through 93.62, specify a minimum of 10.75 and a maximum of 93.63. For integer values, if the desired range is 1 through 10, specify a minimum of 1 and a maximum of 11.

{SEQUENCE from to increment}

The SEQUENCE keyword causes a fixed set of numbers to be generated for the field you specified in the field parameter. The set of numbers begins with the from value and is incremented for each record by the increment value until the to value is equaled or exceeded. The sequence is repeated beginning with the from value. Valid values for from, to, and increment are actual numeric values or the name of a field containing the value.

To generate a decreasing set of numbers, specify a from value greater than the to value, and code a negative increment.

```
{CONSTANT 'value1,value2,. . .,valuen'}
```

The CONSTANT keyword causes a specified series of values to be generated for the field you specified in the field parameter. The sequence is repeated until a value has been generated for each record being created.

Separate the values in the CONSTANT string by commas, and enclose the string in single quotation marks. Valid values consist of actual numeric values only. The entire length of the literal portion of CONSTANT is limited to 40 characters, including commas.

## Operation - Inline

Use the NUMGEN routine only after you have specified the FILEGEN routine. NUMGEN can be specified with ALPHAGEN, DATEGEN, and BADGEN. Conditional execution of NUMGEN is discussed in [Example 3 - Conditional Execution of Data Generation Routines](#) following the BADGEN routine.

## Operation - Database

NUMGEN, with FILEGEN, cannot be used in a database application.

## Examples

The following example illustrates how NUMGEN is used with FILEGEN:

```
FILE CENTFILE F(24)
  NAME      1 12 A
  EMP#     13  5 N
  BIRTH    18  6 N
*
JOB INPUT NULL
%FILEGEN CENTFILE 25 5 NOHEX
%ALPHAGEN NAME ' CUST ' CONSTANT ' JOHNSON, SMITH, PETERS '
%NUMGEN EMP# SEQUENCE 1 10050 1
%DATEGEN BIRTH MMDDYY 0 BETWEEN 010151 010175
```

The following examples demonstrate the three uses of NUMGEN. An example of the full facilities of the test data generation routines is found following the BADGEN routine.

```
BETWEEN
```

This example generates a random value between 3 and 54 for each record being created and writes it to the ZONE field.

```
%NUMGEN ZONE BETWEEN 3 55
SEQUENCE
```

A value of 80 is written in the DEPT field of the first record, 75 to the second record, 70 to the third record, and so on until the value 10 is reached. This sequence is repeated beginning with the value 80.

```
%NUMGEN DEPT SEQUENCE 80 10 -5
```

## CONSTANT

The value 1 is written in the CODE field of the first record, 3 to the second record, 5 to the third record, and so on until the value 8 is reached. This sequence is repeated beginning with the value 1.

```
%NUMGEN CODE CONSTANT '1,3,5,7,9,2,4,6,8'
```

## NUMTEST

The NUMTEST routine evaluates the contents of a specified field for valid numeric data. For each record with non-numeric data in the specified field, an identifying field with a description is printed followed by the hexadecimal representation of the non-numeric field. If the field contains valid numeric data, the field NUMTEST-FLAG is set to the value YES. If the field does not contain valid numeric data, the NUMTEST-FLAG is set to the value NO.

### Syntax

```
%NUMTEST field 'descrip' idfield
```

field

Specify the name of the field being tested. Valid names include any previously defined field.

'descrip'

Code a literal description. This is printed for each non-numeric field value encountered. The maximum length of this parameter is limited by the maximum length of a report line defined at installation. As long as the description is enclosed in single quotation marks, there is no restriction on its content. If you do not enclose the description in quotes, it must not contain any blank characters.

idfield

Specify the name of the ID field. This field is printed on the listing to identify the record with the non-numeric value. Valid names include any previously defined field. Probable choices for the ID field might be account number or invoice number.

## Operation - Inline

For each non-numeric value of field, NUMTEST prints the ID field followed by the description and the hexadecimal representation of the field. If the field contains valid numeric data, an internal field NUMTEST-FLAG is set to the value YES. If the field does not contain valid numeric data, NUMTEST-FLAG is set to the value NO.

## Operation - Database

No change in the specification of parameters is required to use NUMTEST with database files.

## Example

The following is an example of NUMTEST.

Input

```
FILE ...
  ACTNO      1  5  N
  BALANCE    6  7  N  2
  ...
JOB ...
%NUMTEST BALANCE 'FIELD NOT NUMERIC' ACTNO
  ...
```

Output

For each record with a non-numeric field, the first line of output contains the ID field (in this case the account number) and the description. This is followed by a hexprint of the non-numeric BALANCE field.

```
10683 FIELD NOT NUMERIC
```

```
CHAR 07R6739
ZONE FFDFFFF
NUMR 0796739
  1...5..
```

```
11919 FIELD NOT NUMERIC
```

```
CHAR FFFFFFFF
ZONE CCCCCC
NUMR 6666666
  1...5..
```

```
20109 FIELD NOT NUMERIC
```

```
CHAR 0034455
ZONE DFFFFFFF
NUMR 6634455
  1...5..
```

## RANDOM

The RANDOM routine uses a pseudo random number generator to produce a series of random numbers from one to 15 digits in length. The same seed will generate the same series of random numbers. Generation is initiated by a seed parameter, with different seed values resulting in different series of random numbers. The seed is used only for the first pass through the generator.

### Syntax

```
%RANDOM field seed length
```

field

Specify the name of the field to which each number is placed after it is generated. A valid field is any previously defined field with a data type of numeric (N) or alphanumeric (A) and with length as specified in the length parameter.

seed

Specify an arbitrary number, which initiates the random number generator. The seed value is related to a set of random numbers in that the same seed always produces the same set of numbers. A valid value is an actual numeric value or the name of a field containing a numeric value. Values can be up to seven digits in length with no decimal places. Values greater than seven digits are truncated on the left.

length

Length is a numeric value from 1 to 15 that determines how many digits each random number will contain. The number specified for the length parameter must be the same as the length of the field parameter. A valid value for length is an actual numeric value or the name of a field containing a numeric value.

### Operation - Inline

RANDOM generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

### Operation - Database

No change in the specification of parameters is required to use RANDOM with database files.

## Example

The following is an example of RANDOM.

Input

```

...
DEFINE RANDOM-NUMBER      W  8  N
DEFINE COUNTER            W  8  N
...
JOB INPUT NULL
DO WHILE COUNTER LT 10
  %RANDOM RANDOM-NUMBER 47 8
  COUNTER = COUNTER + 1
  DISPLAY +10 RANDOM-NUMBER
END-DO
STOP

```

Output

This example invokes the random number generator in a DO loop to write 10 eight-digit random numbers to the field named RANDOM-NUMBER. The CA-Easytrieve Plus DISPLAY statement is used to create the listing. The seed used to initiate the generation of random numbers is 47.

```

27818477
48722028
73698236
88214818
58375141
28536615
12956286
95804208
65886354
64576492

```

## RANDSPAN

The RANDSPAN routine uses a pseudo random number generator to produce a series of random numbers in a specified range. Generation is initiated by a seed parameter, with different seed values resulting in different series of random numbers. The same seed generates the same series of random numbers. The seed is used only for the first pass through the generator.

### Syntax

```
%RANDSPAN  field  seed  minimum  maximum
```

field

Specify the name of the field to hold the generated number. A valid field is any previously defined numeric (N) or alphanumeric (A) field. The length of the field must be equal to the length of the number defined for the maximum parameter.

seed

Specify an arbitrary number that initiates the random number generator. The same seed always produces the same set of numbers. A valid value is an actual numeric value or the name of a field containing a numeric value. Values can be up to seven digits in length with no decimal places. Values greater than seven digits are truncated on the left.

minimum maximum

Specify a range of numbers to be generated by the random number generator. Valid values for minimum and maximum are actual numeric values or the name of a field containing the value. Values must be no more than 15 digits in length.

Values generated are greater than or equal to the minimum or less than or equal to the maximum. For example, if the desired range is 1 through 10, specify a minimum of 1 and a maximum of 10.

## Operation - Inline

RANDSPAN generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

No change in specification of parameters is required to use RANDSPAN with database files.

## Example

This example invokes the random number generator in a DO loop to write 10 random numbers in the range of 1 through 1,000 to the field named RANDOM-NUMBER. The CA-Easytrieve Plus DISPLAY statement is used to create the listing. The seed used to initiate the random number generator is 999.

Input

```
...  
DEFINE RANDOM-NUMBER W 4 N  
DEFINE COUNTER W 2 N  
...  
JOB INPUT NULL  
DO WHILE COUNTER LT 10  
  %RANDSPAN RANDOM-NUMBER 999 1 1000  
  COUNTER = COUNTER + 1  
  DISPLAY RANDOM-NUMBER  
END-DO  
STOP
```

Output

```
0058
0666
0647
0355
0767
0220
0002
0890
0936
0814
```

## SQRT

The SQRT routine calculates the square root of a specified number. The result is accurate to two decimal places.

### Syntax

```
%SQRT number result
```

number

Specify the numeric value on which the square root calculation is to be performed. A valid value for the number parameter is an actual numeric value or the name of a field containing a numeric value. Values can contain up to two decimal places and are truncated on the right if more than two are specified.

result

Specify the name of the field to which the result of the calculation is placed. A valid name is any previously defined numeric field.

### Operation - Inline

SQRT generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

### Operation - Database

No change in the specification of parameters is required to use SQRT with database files.

## Example

The following is an example of SQRT. This example demonstrates the use of SQRT to calculate the radius for a circle of a given area. The formula for radius given the area is:

$$\text{RADIUS} = \text{SQRT}(\text{AREA}/\text{PI}).$$

Input

```
FILE ...
  AREA          1  8  P  2
  RADIUS        W  6  P  2
JOB ...
DEFINE WORKAREA W  8  P  2
DEFINE PI       W  3  P  4  VALUE (3.1416)
WORKAREA = AREA / PI
%SQRT WORKAREA RADIUS
PRINT RADIUS-REPORT
...
REPORT RADIUS-REPORT LINESIZE 72
TITLE 1 'CALCULATION OF RADIUS'
LINE AREA RADIUS
```

Output

The radius is calculated for each area in the input file, and the area and radius are printed in the RADIUS-REPORT. The field named WORKAREA is used to hold the value of (AREA/PI) for the square root calculation.

```
          CALCULATION OF RADIUS
          AREA          RADIUS
2,345,154.00          863.99
  35,355.78          106.08
   1,968.92           25.03
          .
          .
          .
          400.00          11.28
```

## SRCECOMP

The SRCECOMP routine compares two versions of a source program to determine the differences between them. It prints a listing of the programs or a report of all added, deleted, moved, and changed statements.

## Syntax

```
%SRCECOMP oldfile oldfield newfile newfield {ALL }
                                               {CHANGES}
```

oldfile

Specify the name of the file containing the old source statements to be compared. A valid name is any previously defined file.

oldfield

Specify the name of a field defined in the file identified by oldfile. The location and attributes of this field determine where the comparison will begin for the source statements in the old file and the length of the comparison.

A valid name for oldfield is any field defined in oldfile. It must have a length attribute of less than 255 and the same length attribute as newfield. The starting location of the field can differ from that of newfield.

newfile

Specify the name of the file containing the new source statements to be compared. A valid name is any previously defined file.

newfield

Specify the name of a field defined within the file indicated by newfile. The location and attributes of this field determine where the comparison will begin for the source statements in the new file and the length of the comparison.

A valid name for newfield is any field defined in newfile. It must have a length attribute of less than 255 and the same length attribute as oldfield. The starting location of the field can differ from that of oldfield.

{ALL }  
{CHANGES }

This parameter specifies what will be printed in the SRCECOMP report.

**ALL**—Indicates that both changed and unchanged statements will be listed.

**CHANGES**—Indicates that only changed statements will be listed.

## Operation - Stand-alone REPORT

The SRCECOMP routine can compare a maximum combination of 32,000 identical, added, or deleted statements in the two files. It assigns sequence numbers to statements in the programs being compared. On the report, these numbers appear under the columns OLD SEQ NUM (for statements that appear in the old version) and NEW SEQ NUM (for statements that appear in the new version). These represent the statement's relative position in each file. The maximum length of a record in oldfile or newfile is 6136.

The report lists a maximum of 105 source characters. If the source statements contain more than 105 characters, the listing prints only the first 105 characters of the statement.

Screening of input data in SRCECOMP is not allowed. If screening is required, code the logic to screen an input file in another job step and write the desired records to a temporary file. Then use this temporary file as input to SRCECOMP.

SRCECOMP requires 400 KB of storage for execution.

## Operation - Database

SRCECOMP cannot be used in a database application.

## Example

The following is an example of SRCECOMP.

Input

```
FILE OLDFILE ...
  COMPARE-OLD ...
FILE NEWFILE ...
  COMPARE-NEW ...
...
%SRCECOMP OLDFILE COMPARE-OLD NEWFILE COMPARE-NEW CHANGES
```

Output

This report was generated by the CHANGES option of SRCECOMP and lists only the statements that have been changed. It lists the statement number; whether it has been added, deleted, or moved; where it has been moved from or to; and the first 105 bytes of the statement.

```
4/20/88          SOURCE COMPARE          PAGE          1

OLD   NEW
SEQ   SEQ
NUM   NUM
SOURCE

2      DELETED          PAYDEPT 29 2 N
2      ADDED           SORTFIELD 29 2 N
7      MVD FROM        8 ZONE 27 2 N
8      MVD TO          7 ZONE 27 2 N
10     MVD FROM        12 DEFINE INTAMT W 6 P 2
12     MVD TO          10 DEFINE INTAMT W 6 P 2
15     DELETED         DEFINE FIRSTSW W 1 N VALUE (0)
15     ADDED           DEFINE SWITCH W 1 N VALUE (0)
17     DELETED         SKIP 1
18     DELETED         SORT PAYFILE TO VFMFILE USING (PAYDEPT)
17     ADDED           SORT PAYFILE TO VFMFILE USING (SORTFIELD)
```

```

24      DELETED          IF FIRSTSW EQ 1
23      ADDED           IF SWITCH EQ 1
27      DELETED          FIRSTSW = 1
32      ADDED           SWITCH = 1
35      DELETED          OLDDEPT = DEPT
36      MVD TO          39  INDICATOR = 2
34      ADDED           INTAMT = INTAMT + PAY
35      MVD FROM        40  INDICATOR = 1
40      MVD TO          35  INDICATOR = 1
39      MVD FROM        36  INDICATOR = 2

```

## STDDEV

The STDDEV routine is used to calculate the standard deviation of a set of numbers.

Standard deviation is a statistical value that gives a measurement of the variability of a set of numbers. The greater the variability, the larger the standard deviation.

In most applications for standard deviation, the file is divided into groups or intervals based on the value of a control field. The standard deviation of these intervals and the standard deviation of the entire population are used for statistical purposes and in other Toolkit routines.

To calculate the standard deviation, you:

1. Obtain the difference between the value of each item in the population and the population mean.
2. Square the difference.
3. Add them together.
4. Divide the sum by the total number of items.
5. Extract the square root.

Standard deviation is a required input to the Toolkit statistical routines VARSAMP and VARPCT, and can be obtained for a variety of other statistical purposes.

## Syntax

```
%STDDEV field indicator intervaldev totaldev
```

**field**

Specify the name of the quantitative field for which the standard deviation is calculated. A valid field name is any previously defined quantitative field. Values can contain up to two decimal places and are truncated on the right if more than two are specified.

**indicator**

The indicator parameter is used to inform STDDEV when an input value represents the start of a new interval or when it is just another item in the interval. The method used is as follows.

For indicator, you specify a one-character work field, which is used to indicate three different calculation options to STDDEV (see Options 1, 2, and 3). These options work with the following two parameters to regulate the initialization of internal work fields.

Any job that uses STDDEV must contain logic to control the process of calculating the standard deviation of intervals in a file and/or the standard deviation of the entire population. Intervals must be sequenced; see [Operation - Inline](#).

In some situations, you may not want the population standard deviation to reflect the values from all intervals. In this case, use the indicator parameter to reset to zero the internal work fields for both the interval and total standard deviation.

The following values for indicator control the calculations for the intervaldev and totaldev parameters.

**Option 1 - Initializing the Work Fields (0)**

Set the indicator field to zero when you want to reset both the totaldev and intervaldev fields to zero.

**Option 2 - Calculating Intervals (1)**

Set the indicator field to one at the beginning of each new interval. This initializes the intervaldev field to zero and begins the calculation for the new interval mean.

**Option 3 - Calculating Intervals and Whole Populations (2)**

Set the indicator field to two for the second and all subsequent records in that interval. This causes the STDDEV routine to update the values for intervaldev and totaldev.

intervaldev

Specify the name of a user-defined, quantitative, W-type work field with two decimal places. The standard deviation for the interval is maintained in this field. (Working storage is discussed in the *CA-Easytrieve Plus Reference Guide*.)

When the first record for an interval is detected (when the indicator is set to one), the STDDEV routine initializes intervaldev to zero. When the second and all subsequent records for an interval are processed (when the indicator is two), intervaldev is updated to reflect the standard deviation of the current field parameter value plus the values from all previously processed records in that interval.

totaldev

Specify the name of a user-defined, quantitative, W-type work field with two decimal places. Standard deviation for the entire population is maintained in this field.

Each time STDDEV processes a record, totaldev is updated to reflect the standard deviation of the current field parameter value plus the values of all previous records, unless the indicator parameter has been set to zero. If the indicator parameter has been set to zero, totaldev reflects the standard deviation of all values occurring after the indicator is set to a nonzero value.

## Operation - Inline

If you want interval standard deviation processing, records input to STDDEV must be in sequence according to what comprises the intervals. For example, if each department comprises an interval, records must be sorted according to department. If each interval is a range of dollar amounts, records must be sorted according to those dollar amounts. Sequencing in CA-Easytrieve Plus is performed by the SORT statement. For additional information, see the *CA-Easytrieve Plus Reference Guide*.

## Operation - Database

No change in the specification of parameters is required to use STDDEV with database files.

## Examples

The following two examples are invocations of STDDEV which demonstrate different uses of the indicator parameter.

### Example One

In this example, the standard deviation is determined for intervals of the field PAY based on the control field DEPT.

The following is a brief description of the important fields defined in this example:

- INTAMT-The interval total of the field for which the standard deviation is being calculated
- TOTAMT-The field total for the whole file
- INTDEV-The standard deviation of each interval
- TOTDEV-The standard deviation for the entire file
- OLDDEPT-A working storage field that contains the control value of the last interval
- FIRSTSW-A working storage field that contains the value zero for the first record processed and the value one for all subsequent records
- INDICATOR-The indicator parameter

Input

```

FILE PAYFILE FB(20 2000)
  SORTFIELD      7 3 N
FILE VFMFILE FB(20 2000) VIRTUAL
  PAY           1 6 P 2
  DEPT          7 3 N
  DEFINE INTAMT W 8 P 2. DEFINE TOTAMT W 8 P 2
  DEFINE INTDEV W 8 P 2. DEFINE TOTDEV W 8 P 2
  DEFINE OLDDEPT W 3 N VALUE (0)
  DEFINE FIRSTSW W 1 N VALUE (0)
  DEFINE INDICATOR S 1 N
SORT PAYFILE TO VFMFILE USING (SORTFIELD)
JOB INPUT VFMFILE FINISH END-OF-JOB
  IF PAY NOT NUMERIC
    GO TO JOB
  END-IF
  IF DEPT NE OLDDEPT
    IF FIRSTSW EQ 1
      PRINT STDDEV-REPORT
    END-IF
    PERFORM NEW-INTERVAL
  ELSE
    PERFORM OLD-INTERVAL
  END-IF
  PERFORM STANDARD-DEVIATION
NEW-INTERVAL. PROC

```

```

FIRSTSW = 1
INTAMT = PAY
OLDDEPT = DEPT
INDICATOR = 1
END-PROC
OLD-INTERVAL. PROC
INTAMT = INTAMT + PAY
INDICATOR = 2
END-PROC
STANDARD-DEVIATION. PROC
TOTAMT = TOTAMT + PAY
%STDDEV PAY INDICATOR INTDEV TOTDEV
END-PROC
END-OF-JOB. PROC
PRINT STDDEV-REPORT
END-PROC
REPORT STDDEV-REPORT
TITLE 1 'STANDARD DEVIATION OF PAYFILE BY DEPARTMENT'
HEADING OLDDEPT ('DEPT. ')
HEADING INTAMT ('TOTAL', 'FOR DEPARTMENT')
HEADING INTDEV ('STD. DEVIATION', 'BY DEPARTMENT')
HEADING TOTAMT ('TOTAL', 'FOR COMPANY')
HEADING TOTDEV ('STD. DEVIATION', 'FOR COMPANY')
LINE OLDDEPT INTAMT INTDEV TOTAMT TOTDEV

```

Output

STANDARD DEVIATION OF PAYFILE BY DEPARTMENT

DEPT.	TOTAL FOR DEPARTMENT	STD. DEVIATION BY DEPARTMENT	TOTAL FOR COMPANY	STD. DEVIATION FOR COMPANY
01	6,978,481.91	54,559.37	6,978,481.91	54,559.37
02	7,468,652.90	58,502.32	14,447,134.81	56,667.54
05	7,981,761.60	58,211.22	22,428,896.41	57,401.78
07	7,240,895.64	55,674.23	29,669,792.05	56,984.89
12	6,531,546.62	56,750.47	36,201,338.67	57,119.18
14	7,222,257.51	56,266.94	43,423,596.18	56,979.45
21	7,057,234.07	58,402.64	50,480,830.25	57,187.24

In this example, you begin by defining the library section, and continue by defining all necessary working storage fields. The file PAYFILE is sorted to the CA-Easytrieve Plus virtual file VFMMFILE. (See the CA-Easytrieve Plus *Reference Guide* for a discussion of virtual files.) The sorted VFMMFILE then becomes the input file to the job that computes the standard deviation. The logic is as follows:

- If the PAY field contains a non-numeric value, the GO TO JOB statement bypasses this particular record.
- The input record is read, and the DEPT field is compared to OLDDEPT, which is initialized to zero (it is assumed that no DEPT value of zero exists). If the values are not equal, a new department has been found, and the FIRSTSW field is compared to the value one.

If this is not the first record (FIRSTSW = 1), a line of the report is printed because you have just encountered a new interval (department number). The NEW-INTERVAL procedure is then performed, which sets INDICATOR to one. This tells STDDEV that a new interval has been encountered.

- If a new department is not found, the procedure, OLD-INTERVAL, is performed. This sets INDICATOR to two, which tells STDDEV that the value being processed is in the current interval.
- The STANDARD-DEVIATION procedure is performed, which invokes STDDEV with the correct value for INDICATOR. When the last record has been read, the procedure END-OF-JOB, listed on the JOB statement, prints data for the last interval of the report.

The output lists the total and standard deviation for each department. The running total for the file and the accumulated standard deviation are also listed when each department line is printed. The file total and total standard deviation are the last numbers in the last two columns.

## Example Two

In this example, the standard deviation of the PAY field for the total file is determined. Interval values are not calculated.

The following is a brief description of the important fields defined in this example:

**INTDEV** – A dummy field for the invocation of STDDEV

**TOTAMT** – The field total for the whole file

**TOTDEV** – The standard deviation for the entire file

**INDICATOR** – The indicator parameter

**FIRSTSW** – A working storage field that contains the value zero for the first record processed and the value one for all subsequent records

Input

```

FILE ...
  PAY      1  6  P  2
  DEFINE INTAMT   W  8  P  2.  DEFINE TOTAMT   W  8  P  2
  DEFINE INTDEV   W  8  P  2.  DEFINE TOTDEV   W  8  P  2
  DEFINE INDICATOR S  1  N.  DEFINE FIRSTSW   W  1  N  VALUE (0)
JOB INPUT PAYFILE FINISH END-OF-JOB
  IF PAY NOT NUMERIC
    GO TO JOB
  END-IF
  IF FIRSTSW EQ 0
    INDICATOR = 1
    FIRSTSW = 1
  ELSE
    INDICATOR = 2
  END-IF
  PERFORM STANDARD-DEVIATION
  STANDARD-DEVIATION. PROC
    TOTAMT = TOTAMT + PAY
    %STDDEV PAY INDICATOR INTDEV TOTDEV
  END-PROC

```

```

END-OF-JOB. PROC
  PRINT STDDEV-REPORT
END-PROC
REPORT STDDEV-REPORT
  TITLE 1 'STANDARD DEVIATION OF PAYFILE'
  HEADING TOTAMT ('TOTAL', 'FOR COMPANY')
  HEADING TOTDEV ('STD. DEVIATION', 'FOR COMPANY')
  LINE TOTAMT TOTDEV

```

## Output

```

STANDARD DEVIATION OF PAYFILE

          TOTAL          STD. DEVIATION
          FOR COMPANY    FOR COMPANY
          -----          -----
          49,760,795.68    57,936.50

```

As shown, you begin by defining the library section and continue with the definition of all necessary working storage fields.

The INTDEV field is not used in the execution of the job, but it still must be defined as a working storage field and listed as a parameter on the invocation for STDDEV. Because there is no interval processing, you need not sort the input file. The logic is as follows:

- If the PAY field contains a non-numeric value, the GO TO JOB statement bypasses this particular record.
- If the FIRSTSW field indicates that this is the first record, INDICATOR is set to one. For all other records, INDICATOR is set to two.
- The STANDARD-DEVIATION procedure is performed, which invokes the STDDEV routine.
- When the last record has been read, the procedure END-OF-JOB, listed on the JOB statement, prints the STDDEV-REPORT.

Because this example is run against the same file as Example One, the totals printed in this report are identical to the final totals in the last two columns from Example One.

## TIMECONV

The TIMECONV routine converts time represented in hundredths of a second to a representation including hours, minutes, seconds, and hundredths of seconds.

### Syntax

```
%TIMECONV time1 time2
```

time1

Specify the name of the field to be converted. This field must contain time represented in hundredths of seconds. A valid name is any previously defined numeric field.

time2

Specify the name of the field to which the result of the time conversion will be placed. Results are placed in this field in the format HHMMSShh. A valid name is any previously defined numeric field that will hold eight characters.

## Operation - Inline

TIMECONV generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

No change in the specification of parameters is required to use TIMECONV with database files.

## Example

The following is an example of TIMECONV.

Input

```
FILE INFILE CARD
  TIMEIN 1 8 N 0
  TIMEOUT W 8 N MASK 'ZZ:ZZ:ZZ.99'
  HOURS TIMEOUT 2 N
  MINUTES TIMEOUT +2 2 N
  SECONDS TIMEOUT +4 2 N
  TENTHS TIMEOUT +6 2 N
JOB INPUT INFILE
  %TIMECONV TIMEIN TIMEOUT
  PRINT RPT1
REPORT RPT1 LINESIZE 72
  LINE TIMEIN TIMEOUT HOURS MINUTES SECONDS TENTHS
END
00000001
00000010
00000100
00001234
00060000
00120000
00240000
10601234
/*
```

## Output

The TIMEIN field is converted from hundredths of a second to hours, minutes, seconds, and hundredths of a second. After the calculation is made, the converted time is placed in the TIMEOUT field.

TIMEIN	TIMEOUT	HOURS	MINUTES	SECONDS	TENTHS
1	.01	00	00	00	01
10	.10	00	00	00	10
100	1.00	00	00	01	00
1,234	12.34	00	00	12	34
60,000	10:00.00	00	10	00	00
120,000	20:00.00	00	20	00	00
240,000	40:00.00	00	40	00	00
10,601,234	29:26:52.34	29	26	52	34

## UNBYTE

Information is stored and transmitted on a computer's magnetic devices by a two-state data representation. The presence or absence of magnetized spots on the surface of a revolving magnetic disk or reel of magnetic tape represents the value of zero (absence) or value of one (presence) to the computer. Each piece of information is referred to as a binary digit, or bit. A sequence of eight adjacent bits forms one byte.

Because a byte is the smallest addressable unit on the computer, UNBYTE, the bit manipulation routine, makes it easier for you to access encoded information on a bit-for-bit basis. UNBYTE provides a method of investigating all eight bits of a given byte.

For example, a 0 in a certain position of a byte can be used to indicate male, and a 1 can indicate female. The UNBYTE routine takes an input field and creates nine fields, named BIT0, BIT1, BIT2, . . . , BIT7, and ALLBITS. Fields BIT0 through BIT7 are one byte fields that correspond to the value (0 or 1) of bits 0 through 7 of the input byte. ALLBITS is an eight byte field that contains the values of each of the individual fields BIT0 through BIT7.

### Syntax

```
%UNBYTE inputbyte
```

inputbyte

Specify the name of a field of which bits are placed in fields BIT0 through BIT7 and ALLBITS, as discussed previously. A valid name is any previously defined field. The field must be only one byte in length.

## Operation - Inline

The UNBYTE routine can be used as often as required to interrogate as many bytes as necessary. However, each time it executes, the BIT0 through BIT7 and ALLBITS fields are reused. These fields are defined by UNBYTE and must not be defined by you.

Since the INPUTBYTE is restricted to being one byte in length, the CA-Easytrieve Plus overlay redefinition concept can be used to define a one-byte field from a multi-byte field. For details, see the topic on the DEFINE Statement, in the *CA-Easytrieve Plus Reference Guide*.

UNBYTE generates no output and can be used alone or with other routines and/or CA-Easytrieve Plus logic.

## Operation - Database

No change in the specification of parameters is required to use UNBYTE with database files.

## Example

The following is an example of UNBYTE.

Input

```
FILE PAYROLL
  PAY-STATUS      1  1  B
DEFINE MALE-COUNTER  W  3  P  0
DEFINE FEMALE-COUNTER  W  3  P  0
JOB INPUT PAYROLL FINISH END-OF-JOB
%UNBYTE PAY-STATUS
IF BIT4 EQ 0
  MALE-COUNTER = MALE-COUNTER + 1
ELSE
  FEMALE-COUNTER = FEMALE-COUNTER + 1
END-IF
END-OF-JOB. PROC
DISPLAY 'PAYROLL DEPARTMENT REPORT'
DISPLAY SKIP 1, 'NUMBER OF MALES =', MALE-COUNTER
DISPLAY 'NUMBER OF FEMALES =', FEMALE-COUNTER
END-PROC
```

Output

This sample examines bit 4 of the PAY-STATUS byte of a payroll file, counts the number of male and female employees, and prints the totals after all payroll records have been examined.

```
PAYROLL DEPARTMENT REPORT

NUMBER OF MALES   =   118
NUMBER OF FEMALES =    94
```

## WEEKDAY

The WEEKDAY routine calculates the day of the week from a given date. The date can be in any format. WEEKDAY only calculates the day of the week for the years 1940 through 2039. The day of the week calculated is written to a specified field.

### Syntax

```
%WEEKDAY  date  format  day-of-week [THRESHOLD value]
```

date

Specify the name of the field containing the date for which the day of the week is calculated. The date in this field must be in the format specified by FORMAT. A valid name is any previously defined field.

format

Specify the format of the date field. Format is a literal description of pairs of letters. The letters indicate positions as follows:

```
MM = month
DD = day
YY = year
CC = century
```

The value of Date is not checked for a valid date with the specified format. If you want validation, use the DATEVAL routine before using WEEKDAY. The only valid Julian format is YYDDD.

The following are some, but not all, of the valid formats:

```
MMDDYY
MMDDCCYY
YYMMDD
YYDDD (Julian)
```

day-of-week

Specify the name of a previously defined alphanumeric field to which the resulting day of the week is written. The field must be at least nine bytes to avoid truncating the day of the week.

[THRESHOLD value]

The THRESHOLD parameter is used to determine the century value if it is not supplied in the century format (CC) in the date. Specify a value that establishes the upper end of a one-hundred-year range in the 20th and 21st centuries used to control the CC portion of generated dates.

**Note:** Unlike other macros that use the THRESHOLD value, the WEEKDAY routine has a default THRESHOLD value of 39. This provides a range from 1940 to 2039 inclusive. You can override this value.

General rules for specifying THRESHOLD values are:

- The THRESHOLD value is ignored if you provide a century value (CC).
- If the dates to be generated do not exceed the year 2000, specify the THRESHOLD a value of 0. This causes all dates to have a range of 1901 through 2000.
- If the dates exceed the year 2000, choose a THRESHOLD high enough to generate correct dates in the 21st century but not so high as to convert dates from the 20th century to the 21st century.
- When dates to be generated do not involve calculations for century, specify the THRESHOLD default value of 0.
- Valid values for THRESHOLD are 0 through 99.

For example, if THRESHOLD is 40, the upper boundary of the range is set to 2040, and the lower boundary is 1941. When converting YY to CCYY, each year is assigned a two-position century based on the range established by THRESHOLD. In this example, if year is 52, century is 19; if year is 21, century is 20.

It is important that the THRESHOLD value be correct for the range of dates to be generated. For example, if WEEKDAY is invoked to process dates between the years 1949 and 1952, and THRESHOLD is 50, the years 1949 and 1950 become 2049 and 2050, while the years 1951 and 1952 remain 1951 and 1952. In this respect, the YY (year) portion of the date controls the CC (century) portion in accordance with the THRESHOLD value.

## Operation - Inline

WEEKDAY generates no output and can be used with other routines and/or CA-Easytrieve Plus logic. WEEKDAY does no date checking and assumes that the date input is valid.

## Operation - Database

No change in specification of parameters is required to use WEEKDAY with database files.

## Example

In this example, the day of the week is calculated from the date contained in the specified field, and the output is printed using a PRINT statement.

Input

```
FILE INFILE CARD
GREG-DATE 1 6 N HEADING ('GREGORIAN' 'DATE') MASK 'Z9/99/99'
*
DAY-OF-WEEK W 9 A HEADING ('DAY OF' 'WEEK')
*
JOB INPUT INFILE
%WEEKDAY GREG-DATE MMDDYY DAY-OF-WEEK
PRINT REPORT1
REPORT REPORT1 LINESIZE 78
TITLE 1 'TEST OF DAY OF THE WEEK MACRO'
*
LINE GREG-DATE DAY-OF-WEEK
*
END
```

Output

```
4/13/88          TEST OF DAY OF THE WEEK MACRO          PAGE 1
                GREGORIAN    DAY OF
                DATE          WEEK
                11/01/87     SUNDAY
                11/02/87     MONDAY
                11/03/87     TUESDAY
                11/04/87     WEDNESDAY
                11/05/87     THURSDAY
                11/06/87     FRIDAY
                11/07/87     SATURDAY
                12/03/87     THURSDAY
                1/04/87      SATURDAY
                1/05/87      SUNDAY
                2/06/87      THURSDAY
                3/07/87      FRIDAY
                11/08/87     SUNDAY
                11/09/87     MONDAY
                11/10/87     TUESDAY
                11/11/87     WEDNESDAY
                11/12/87     THURSDAY
                11/13/87     FRIDAY
```



# Basic SMF Reporting Facility

The SMF Reporting System provides a simple, yet flexible method to access and report on data from the IBM System Management Facilities (SMF). You can obtain results from the SMF reporting system by invoking macro statements that do not require a detailed knowledge of SMF records. The SMF reporting system provides two methods for accessing SMF data:

- **SMF Audit Routines** – These routines provide an easy-to-use method of accessing SMF data and formatting SMF data into meaningful reports.
- **SMF Record Field Definitions** – These routines provide the flexibility to support customized reports through user-written routines. These routines contain the field definitions for most SMF record types. For additional information, see [SMF Record Field Definitions](#) later in this chapter.

**Note:** All dates are processed in the range 1970 through 2069.

## SMF Audit Routines

The SMF audit routines prepare various reports on the information in certain SMF record types. The following is a list of the SMF audit routine names and their functions:

Routine Name	SMF Record Type Reported On	Function Reported On
SMF000	00	IPLs
SMF004	04	Abnormal step terminations
SMF005	05	Abnormal job terminations
SMF006	06	Control output forms
SMF007	07	SMF lost data
SMF014	14	Data set activity
SMF017	17	Scratched data sets
SMF018	18	Renamed data sets

Routine Name	SMF Record Type Reported On	Function Reported On
SMF020	20	Job initiations
SMF049	49	JES2 and JES3 integrity (OS/390 and above only)
SMF062	62	VSAM opens
SMF067	67	VSAM data set deletes
SMF068	68	VSAM data sets renamed
SMFCNT	All SMF record types	Frequency distribution of record types showing record collection activities

**Note:** Detailed information on each routine is provided in later sections of this chapter.

SMF audit routines follow a naming convention of SMF0xx, where xx is the number of the SMF record type on which the SMF audit routine makes the report. The SMFCNT routine reports on all record types in a frequency distribution report. You can run SMFCNT first to determine the number and percentage of types of records that are on the SMF file.

You can invoke the SMF audit routines in a manner similar to stand-alone REPORT Toolkit routines. Each routine consists of two macro invocation statements and is invoked through the passing of parameters. In addition, you can stack routines to produce multiple reports from a single job step. You can also stack the same routine within one job.

By stacking routines you can generate multiple reports in one pass of the file. This increases the efficiency and flexibility of the reporting system. Each routine is separated into two parts:

- The first part performs the logic to collect the necessary data from the SMF file.
- The second part consists of the report to generate the listing.

To create the listing that you want, you must invoke both parts of a routine with the appropriate parameters.

The following example demonstrates the use of the SMF reporting system:

```
%SMFILE
%SMF014A DATE '''840101''' '''840131'''
%SMF014A JOBNAME '''A''' '''Z9999999''' REPORT J14#2
%SMF017A 000000 240000 840101 840131
%SMF018A 000000 240000 840101 840131
%SMF014B SORT TIME CONTROL DATE
%SMF014B SORT 'DSNAME TIME' CONTROL DATE REPORT J14#2
%SMF018B SORT 'DATE D'
%SMF017B SORT DATE
```

The first invocation is for the macro SMFILE. This macro contains the necessary file and field definitions to access the SMF data file. When using SMF audit routines, the first macro that you invoke must always be SMFILE. SMFILE must also be the DDNAME of the DD statement in the JCL referring to the SMF data file.

**Note:** For SMF files generated by MVS/XA 2.2.0, replace %SMFILE with %S22FILE to access the SMF data file. Likewise, for SMF files generated by MVS/ESA 3.1.3, replace %SMFILE with %SE13FILE to access the SMF data file.

This is followed by the invocations for the SMF routines. The first two invocations are for SMF014 reports. The selection criterion for the first report is the DATE field, and for the second report, the JOBNAME field. The first SMF014A routine is associated with the first SMF014B routine, which specifies sorting by the TIME field. The second SMF014A routine is associated with the second SMF014B routine by specifying J14#2 in the REPORT parameter of both SMF014A and SMF014B. When the same routine is invoked more than once in a single execution, the REPORT parameter associates the A and B routines.

Next, the SMF017 and SMF018 routines are invoked. Notice that SMF017A is invoked before SMF018A, while SMF018B is invoked before SMF017B. The D following the field DATE in the SORT parameter of SMF018B specifies the sorting of records in descending sequence.

The rules for specifying the A and B routines are:

- All A routines must precede all B routines.
- The reports generate in the order of specification of the B routines.
- The REPORT parameter associates the processing logic (A routine) with the actual report (B routine).
- You can insert screening code between the SMFILE statement and the first A routine. However, screening applies to all reports in the processing stream.

## SMF Record Field Definitions

These routines provide field definitions for the majority of SMF record types. The field names follow the conventions listed in the IBM guide *OS/390 MVS System Management Facilities (SMF)*. These routines provide the field definitions for customized SMF reports that you write in CA-Easytrieve Plus. These routines eliminate the time-consuming task of coding field definitions for the various SMF record types.

The following example illustrates accessing an SMF data file:

```
FILE SMFDATA ...  
%SMFR20  
...user written code ....
```

The SMFR20 routine provides the definitions for the various fields of a pre-MVS 2.2.0 SMF Type 20 record. You can then write CA-Easytrieve Plus statements that refer to any field in a type 20 SMF record. Since many fields in an SMF data record are variable in length, indexing methods are used to access these fields. Later in this chapter, you can find a description of the operation of each SMF record type.

## IPL Reporting - SMF000

The SMF000 routine creates a report listing each occurrence of an Initial Program Load (IPL) by the CPU, the time and date of each IPL, and the system options in effect. The parameters that you pass allow for selection based on date and time.

### Syntax

```
%SMF000A starttime endtime startdate enddate [REPORT reportname]  
%SMF000B [REPORT reportname]
```

starttime

Specify the time that the routine will begin reporting on system IPLs. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on system IPLs. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on system IPLs. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on system IPLs. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical report name must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

## Operation

In order to include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF000.

Input

```
%SMFILE
%SMF000A 000000 240000 850318 850323
%SMF000B
```

Output

SMF000 - INITIAL PROGRAM LOAD FOR SYSTEM 3083

IPL DATE	IPL TIME	SMF ACCOUNT REQUESTS	NUMBER OF IPLS
85/03/19	17:05:16	SYSTEM.JOB SYSTEM.JOB.STEP USER EXITS DATA SET VOLUME TEMP DATA SET	
85/03/20	1:53:22	SYSTEM.JOB SYSTEM.JOB.STEP USER EXITS DATA SET VOLUME TEMP DATA SET	
85/03/22	5:26:30	SYSTEM.JOB SYSTEM.JOB.STEP USER EXITS	

DATA SET  
VOLUME  
TEMP DATA SET

FINAL TOTAL

3

This report lists all IPL reports in the SMF data set.

## Abnormal Step Termination - SMF004

The SMF004 routine creates a report of abnormally terminated steps and computes the elapsed time for the step. The parameters that you pass allow for selection, sorting, and totaling based on DATE, TIME, JOBNAME, PROGRAM\_, USERID\_, or ABEND.

### Syntax

```
%SMF004A selection      {selectstart selectend
                        {'STRING' 'value1,value2,...valuen'}} +
[REPORT reportname]

%SMF004B [REPORT reportname] [SORT sortflds [D]] +
[CONTROL cntrlflds [options]]
```

Specify the field in the SMF record that determines the range for selection of records in this report. The specification of this parameter determines the valid values for the selectstart and selectend parameters. The value that you specify for selection must be one of the following:

**DATE**—Date determines selection. Selectstart, selectend, or the individual values that STRING specifies must be dates in YYMMDD format. Valid values for selectstart and selectend are a six-byte alphanumeric field or actual six-digit numeric values enclosed in triple quotes.

If using the STRING option, the keyword STRING and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual six-digit numeric values.

**TIME**—Time determines selection. Selectstart, selectend, or the individual values that STRING specifies must be times in HHMMSS format. Valid values for selectstart and selectend are a six-byte alphanumeric field or actual six-digit numeric values enclosed in triple quotes.

If using the STRING option, the keyword STRING and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual six-digit numeric values.

**JOBNAME**—Job name determines selection. Selectstart, selectend, or the individual values that STRING specifies must be alphabetic literals representing job names. Valid values for selectstart and selectend are an eight-byte alphanumeric field or a character string up to eight characters long enclosed in triple quotes.

If using the **STRING** option, the keyword **STRING** and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual alphanumeric values, each containing a character string up to eight characters long.

**USERID\_** – User ID determines selection. **Selectstart**, **selectend**, or the individual values that **STRING** specifies must be alphabetic literals representing user IDs. Valid values for **selectstart** and **selectend** are an eight-byte alphanumeric field or a character string up to eight characters long enclosed in triple quotes.

If using the **STRING** option, the keyword **STRING** and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual alphanumeric values, each containing a character string up to eight characters long.

**PROGRAM\_** – Program name determines selection. **Selectstart**, **selectend**, or the individual values that **STRING** specifies must be alphabetic literals representing program names. Valid values for **selectstart** and **selectend** are an eight-byte alphanumeric field or character string up to eight characters long enclosed in triple quotes.

If using the **STRING** option, the keyword **STRING** and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual alphanumeric values, each containing a character string up to eight characters long.

**ABEND** – Abend code determines selection. **Selectstart**, **selectend**, or the individual values that **STRING** specifies must be alphabetic literals representing abend codes. You must define **selectstart** and **selectend** as two-byte binary fields containing valid abend values, such as 00C4, 00C7, and so forth. Numeric values for **selectstart** and **selectend** are not allowed when **ABEND** is specified in the selection parameter.

If using the **STRING** option, the keyword **STRING** and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. The values must be actual alphanumeric values representing valid four-digit hexadecimal numbers.

```
{selectstart selectend
{' 'STRING' ' 'value1,value2,...valuen' ' }
```

Specify whether selection is to be by starting and ending points or by a series of individual values.

**selectstart**

Specify the beginning of the selection range for the field chosen in the selection parameter. Valid values vary depending on the field chosen for the selection parameter.

selectend

Specify the end of the selection range for the field chosen in the selection parameter. Valid values vary depending on the field chosen for the selection parameter.

STRING

Specify a series of individual values, separated by commas. The keyword STRING and the string of individual values must each be enclosed in triple quotes. The total length of the string, including commas and triple quotes, must not be greater than 254 characters.

### Examples

```
%SMF004A DATE '''STRING''' '''860612,861130,870101,871231,880101'''  
%SMF004A ABEND '''STRING''' '''0322,00C4,00C7,0806'''  
%SMF004A JOBNAME '''STRING''' '''YEAREND,ARDEPT03,PAYROLL1,AUDIT'''
```

These values are the individual values that are reported on for the particular select parameter. Valid values vary depending upon the field chosen for the selection parameter.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

[SORT sortflds [D]]

Specify the field or fields that you want to use to sequence the report. Valid values are any of the field names described in the selection parameter. To specify multiple fields for sequencing, enclose the field names in single quotes and separate the field names by one blank space.

To specify a descending sort sequence, insert the letter D after the appropriate field name. You must use a blank space to separate the D from the previous and any subsequent field names. When you specify descending sequence, enclose all items in the parameter list in single quotes.

[CONTROL cntrlflds [options]]

Specify the field or fields that you want for control breaks in the reporting process. Valid values for cntrlflds are any of the field names described in the selection parameter. To specify multiple fields for control breaks, enclose the field names in single quotes and separate them by one blank space.

You can specify report processing options after the control break field. These options customize the report in relation to control break activities. The options are:

**NEWPAGE**—Causes a skip to top-of-page after processing is complete for the control break field.

**RENUM**—Performs the same function as NEWPAGE and also resets the page number to one following the control break.

**NOPRINT**—Suppresses printing of the summary line group for the control break that you specify.

If you specify any of these options, separate all items in the parameter list by one blank space and enclose the entire string in single quotes.

## Operation

To include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Examples

The following are two examples of SMF004.

Input (Example 1)

```
%SMFILE
DEFINE ABEND-SELECT W 2 B VALUE('0322')
%SMF004A ABEND ABEND-SELECT ABEND-SELECT
%SMF004B SORT 'PROGRAM_ DATE TIME' CONTROL PROGRAM
```

Output (Example 1)

```
SMF004 - ABNORMAL STEP TERMINATION REPORT                                PAGE      1
                                SYSTEM ID 3081
                                SELECTED BY ABEND
                                SEQUENCED BY SYSID PROGRAM DATE TIME
COMPLETION ABEND
CODE        TYPE  JOBNAME  DATE    TIME  USERID  PROGRAM  MINUTES
                                ELAPSED
                                TALLY
      0322  SYSTEM  JONES   89/12/19 15:40:40-  JIFSEL  11.3685
PROGRAM TOTAL                                11.3685      1
SYSID TOTAL                                11.3685      1
FINAL TOTAL                                11.3685      1
TOTAL TYPE 4 RECORDS                        1,439
TOTAL SELECTED                              1
PERCENT                                     .06
```

This report lists all 0322 abnormal terminations in the SMF data set. The desired ABEND code must be passed to the routine in a previously defined field. ABEND-SELECT is defined as a two-byte binary field containing the value '0322' and is specified as the SELECTSTART and SELECTEND parameters in the SMF004A invocation statement. The report is sequenced by PROGRAM\_, DATE, and TIME within system id (SYSID).

Input (Example 2)

```
%SMFILE
%SMF004A ABEND '''STRING''' '''00C4,0213,0222'''
%SMF004B SORT 'PROGRAM_ DATE TIME' CONTROL PROGRAM
```

Output (Example 2)

SMF004 - ABNORMAL STEP TERMINATION REPORT								PAGE 1	
SYSTEM ID 3081									
SELECTED BY ABEND									
SEQUENCED BY SYSID PROGRAM DATE TIME									
COMPLETION	ABEND	JOBNAME	DATE	TIME	USERID	PROGRAM	MINUTES	ELAPSED	TALLY
CODE	TYPE						TIME		
0222	SYSTEM	SMITH	89/12/19	14:39:31-		PSIS5320	3.4201		
0222	SYSTEM	SMITH	89/12/19	14:49:32-		PSIS5320	4.5176		
0222	SYSTEM	SMITH	89/12/19	15:50:30-		PSIS5320	3.3060		
PROGRAM TOTAL							11.2437		3
0222	SYSTEM	DBMSPCAT	89/12/19	15:59:27-		FDRDSF	1.1030		
PROGRAM TOTAL							1.1030		1
0222	SYSTEM	JONES	89/12/19	11:27:15-		JIFSEL	5.2191		
PROGRAM TOTAL							5.2191		1
0222	SYSTEM	KINSSCAN	89/12/19	9:00:53-		TAPESCAN	26.7758		
PROGRAM TOTAL							26.7758		1
SYSID TOTAL							44.3416		6
FINAL TOTAL							44.3416		6
TOTAL TYPE 4 RECORDS				1,439					
TOTAL SELECTED				6					
PERCENT				.41					

This sample lists 00C4, 0213 and 0222 abnormal terminations in the SMF data set. The desired ABEND codes are passed to the SMF004A routine by means of the STRING option and a string of abend codes. The report is sequenced by PROGRAM\_, DATE, and TIME within system id (SYSID).

## Abnormal Job Terminations - SMF005

The SMF005 routine creates a report of jobs that terminate abnormally and computes the elapsed time for the job. The parameters that you pass allow for selection, sorting, and totaling based on DATE, TIME, JOBNAME, PROGRAMMER, USERID\_, or ABEND.

### Syntax

```
%SMF005A selection {selectstart selectend
                    {''STRING'' ''value1,value2,...valuen'' } +
[REPORT reportname]
%SMF005B [REPORT reportname] [SORT sortflds [D]] +
          [CONTROL cntrlflds [options]]
```

selection

Specify the field in the SMF record that determines the range for selection of records in this report. The specification of this parameter determines the valid values for the selectstart and selectend parameters. The value that you specify for selection must be one of the following:

**DATE**—Date determines selection. Selectstart, selectend, or the individual values that STRING specifies must be dates in YYMMDD format. Valid values for selectstart and selectend are a six-byte alphanumeric field or actual six-digit numeric values enclosed in triple quotes.

If using the STRING option, the keyword STRING and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual six-digit numeric values.

**TIME**—Time determines selection. Selectstart, selectend, or the individual values that STRING specifies must be times in HHMMSS format. Valid values for selectstart and selectend are a six-byte alphanumeric field or actual six-digit numeric values enclosed in triple quotes.

If using the STRING option, the keyword STRING and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual six-digit numeric values.

**JOBNAME**—Job name determines selection. Selectstart, selectend, or the individual values that STRING specifies must be alphabetic literals representing job names. Valid values for selectstart and selectend are an eight-byte alphanumeric field or a character string up to eight characters long enclosed in triple quotes.

If using the STRING option, the keyword STRING and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual alphanumeric values, each containing a character string up to eight characters long.

**USERID\_**—User ID determines selection. Selectstart, selectend, or the individual values that STRING specifies must be alphabetic literals representing user IDs. Valid values for selectstart and selectend are an eight-byte alphanumeric field or a character string up to eight characters long enclosed in triple quotes.

If using the STRING option, the keyword STRING and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual alphanumeric values, each containing a character string up to eight characters long.

**PROGRAM\_**—Program name determines selection. Selectstart, selectend, or the individual values that STRING specifies must be alphabetic literals representing program names. Valid values for selectstart and selectend are an eight-byte alphanumeric field or character string up to eight characters long enclosed in triple quotes.

If using the STRING option, the keyword STRING and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. Valid values are actual alphanumeric values, each containing a character string up to eight characters long.

**ABEND**—Abend code determines selection. Selectstart, selectend, or the individual values that STRING specifies must be alphabetic literals representing abend codes. You must define selectstart and selectend as two-byte binary fields containing valid abend values, such as 00C4, 00C7, and so on. Numeric values for selectstart and selectend are not allowed when ABEND is specified in the selection parameter.

If using the STRING option, the keyword STRING and the string of individual values must each be enclosed in triple quotes. Separate each individual value in the string with a comma. The values must be actual alphanumeric values representing valid four-digit hexadecimal numbers.

```
{selectstart selectend  
{''STRING'' ''value1,value2,...valuen'' }
```

Specify whether selection is to be by starting and ending points, or by a series of individual values.

selectstart

Specify the beginning of the selection range for the field chosen in the selection parameter. Valid values vary depending on the field chosen for the selection parameter.

selectend

Specify the end of the selection range for the field chosen in the selection parameter. Valid values vary depending on the field chosen for the selection parameter.

**STRING**

Specify a series of individual values, separated by commas. The keyword **STRING** and the string of individual values must each be enclosed in triple quotes. The total length of the string, including commas and triple quotes, must not be greater than 254 characters.

**Examples**

```
%SMF005A DATE '''STRING''' '''860612,861130,870101,871231,880101'''
%SMF005A ABEND '''STRING''' '''0322,00C4,00C7,0806'''
%SMF005A JOBNAME '''STRING''' '''YEAREND,ARDEPT03,PAYROLL1,AUDIT'''
```

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

[SORT sortflds [D]]

Specify the field or fields that you want to use to sequence the report. Valid values are any of the field names described in the selection parameter. To specify multiple fields for sequencing, enclose the field names in single quotes and separate the field names by one blank space.

To specify a descending sort sequence, insert the letter D after the appropriate field name. You must use a blank space to separate the D from the previous and any subsequent field names. When you specify descending sequence, enclose all items in the parameter list in single quotes.

[CONTROL cntrlflds [options]]

Specify the field or fields that you want for control breaks in the reporting process. Valid values for cntrlflds are any of the field names described in the selection parameter. To specify multiple fields for control breaks, enclose the field names in single quotes and separate them by one blank space.

You can specify report processing options after the control break field. These options customize the report in relation to control break activities. These options are:

**NEWPAGE**— Causes a skip to top-of-page after processing is complete for the control break field.

**RENUM**— Performs the same function as **NEWPAGE** and also resets the page number to one following the control break.

**NOPRINT**—Suppresses printing of the summary line group for the control break that you specify.

If you specify any of these options, separate all items in the parameter list by one blank space and enclose the entire string in single quotes.

### Operation

To include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

### Examples

The following are two examples of SMF005.

Input (Example 1)

```
%SMFILE
%SMF005A DATE '''891219''' '''891219'''
%SMF005B SORT 'PROGRAMMER DATE TIME' CONTROL PROGRAMMER
```

Output (Example 1)

SMF005 - ABNORMAL JOB TERMINATION REPORT								PAGE 1	
SYSTEM ID 3081									
SELECTED BY DATE									
SEQUENCED BY SYSID PROGRAMMER DATE TIME									
COMPLETION	ABEND					PROGRAMMER	MINUTES	ACCOUNT	
CODE	TYPE	JOBNAME	DATE	TIME	USERID	NAME	ELAPSED	NUMBER	TALLY
							TIME		
0013	SYSTEM	JOB	89/12/19	7:25:33			.0285		
01F6	USER	ABEL	89/12/19	13:39:02 -			.4793	626B	
0013	SYSTEM	JOB	89/12/19	14:52:57			.0203		
PROGRAMMER TOTAL							.5281		
0222	SYSTEM	SMITH1	89/12/19	14:39:31 -		SMITH	3.7870	606A	
0222	SYSTEM	SMITH1	89/12/19	14:49:32 -		SMITH	5.1045	606A	
0222	SYSTEM	SMITH1	89/12/19	15:50:30 -		SMITH	3.5668	606A	
PROGRAMMER TOTAL							12.4583		
0222	SYSTEM	DBMSPCAT	89/12/19	15:59:27 -		DATA CENTER BKUPS	1.1035	625A	
0378	USER	DBUSR034	89/12/19	22:21:59 -		DATA CENTER BKUPS	28.0801	625A	
0378	USER	DBUSR035	89/12/19	22:37:33 -		DATA CENTER BKUPS	29.7345	625A	
0378	USER	DBUSR051	89/12/19	23:13:53 -		DATA CENTER BKUPS	27.3636	625A	
PROGRAMMER TOTAL							86.2817		
0806	SYSTEM	JOHNSON	89/12/19	10:19:51 -		JIFSEL	.3183	609A	
0222	SYSTEM	JOHNSON	89/12/19	11:27:15 -		JIFSEL	5.2193	609A	
0322	SYSTEM	JOHNSON	89/12/19	15:40:40 -		JIFSEL	11.3690	609A	
PROGRAMMER TOTAL							16.9066		
0222	SYSTEM	JONE	89/12/19	9:00:53 -		JONES	26.7761	630A	

```

PROGRAMMER TOTAL                                26.7761

  0706  SYSTEM  JACKSON1  89/12/19 15:36:50 -    JACKSON    .3045    606A
  00C1  SYSTEM  JACKSON1  89/12/19 17:33:38 -    JACKSON    .1711    606A
  00C1  SYSTEM  JACKSON1  89/12/19 17:35:54 -    JACKSON    .1578    606A
PROGRAMMER TOTAL                                .6334

SYSID TOTAL                                    143.5842
FINAL TOTAL                                    143.5842

TOTAL TYPE 5 RECORDS                          466
TOTAL SELECTED                                17
PERCENT                                        3.64
    
```

This sample lists all abnormal terminations of jobs in the SMF data set. The report is sequenced by PROGRAMMER, DATE, and TIME with control breaks on the field PROGRAMMER within system id (SYSID).

Input (Example 2)

```

%SMFILE
%SMF005A DATE '''STRING''' '''891219'''
%SMF005B SORT 'PROGRAMMER DATE TIME' CONTROL PROGRAMMER
    
```

Output (Example 2)

```

SMF005 - ABNORMAL JOB TERMINATION REPORT
SYSTEM ID 3081
SELECTED BY DATE
SEQUENCED BY SYSID PROGRAMMER DATE TIME
PAGE 1

COMPLETION ABEND          PROGRAMMER          MINUTES
CODE        TYPE  JOBNAME  DATE    TIME  USERID  NAME          ELAPSED
              ACCOUNT
              NUMBER  TALLY

  0013  SYSTEM  JOB      89/12/19 7:25:33          .0285
  01F6  USER   ABEL    89/12/19 13:39:02 -    .4793    626B
  0013  SYSTEM  JOB      89/12/19 14:52:57          .0203
PROGRAMMER TOTAL                                .5281

  0222  SYSTEM  SMITH1  89/12/19 14:39:31 -    SMITH     3.7870    606A
  0222  SYSTEM  SMITH1  89/12/19 14:49:32 -    SMITH     5.1045    606A
  0222  SYSTEM  SMITH1  89/12/19 15:50:30 -    SMITH     3.5668    606A
PROGRAMMER TOTAL                                12.4583

  0222  SYSTEM  DBMSPCAT 89/12/19 15:59:27 -    DATA CENTER BKUPS  1.1035    625A
  0378  USER   DBUSR034 89/12/19 22:21:59 -    DATA CENTER BKUPS 28.0801    625A
  0378  USER   DBUSR035 89/12/19 22:37:33 -    DATA CENTER BKUPS 29.7345    625A
PROGRAMMER TOTAL                                58.9181

  0806  SYSTEM  JOHNSON 89/12/19 10:19:51 -    JIFSEL    .3183    609A
  0222  SYSTEM  JOHNSON 89/12/19 11:27:15 -    JIFSEL    5.2193    609A
  0322  SYSTEM  JOHNSON 89/12/19 15:40:40 -    JIFSEL   11.3690    609A
PROGRAMMER TOTAL                                16.9066

  0222  SYSTEM  JONE    89/12/19 9:00:53 -    JONES     26.7761    630A
PROGRAMMER TOTAL                                26.7761

  0706  SYSTEM  JACKSON1 89/12/19 15:36:50 -    JACKSON    .3045    606A
  00C1  SYSTEM  JACKSON1 89/12/19 17:33:38 -    JACKSON    .1711    606A
  00C1  SYSTEM  JACKSON1 89/12/19 17:35:54 -    JACKSON    .1578    606A
PROGRAMMER TOTAL                                .6334
    
```

SYSID TOTAL		116.2206
FINAL TOTAL		116.2206
TOTAL TYPE 5 RECORDS	466	
TOTAL SELECTED	16	
PERCENT	3.43	

This sample lists all abnormal terminations of jobs in the SMF data set. The date is passed to the SMF005A routine by means of the STRING option. The report is sequenced by PROGRAMMER, DATE, and TIME with control breaks on the field PROGRAMMER within system id (SYSID).

## Control Output Forms - SMF006

The SMF006 routine creates a report listing where printed reports are routed and if any operator intervention occurred. The report is sequenced by TIME, DATE, JOBNAME, and FORM#. Control breaks occur when the FORM# field changes within system id (SYSID). The parameters that you pass allow for selection based on date and time.

### Syntax

```
%SMF006A starttime endtime startdate enddate [REPORT reportname]
%SMF006B [REPORT reportname]
```

starttime

Specify the time that the routine will begin reporting on output forms. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on output forms. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on output forms. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on output forms. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

### Operation

In order to include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

### Example

The following is an example of SMF006.

Input

```
%SMFILE
%SMF006A 000000 240000 891219 891220
%SMF006B
```

Output

SMF006 - OUTPUT WRITER FORMS CONTROL REPORT							PAGE	1
SYSTEM 3081								
FORM BLNK								
JOBNAME	DATE	TIME	FORM NUMBER	RECORD COUNT	ROUTED	OPERATOR INTERVENTION	JOB COUNT	
PR90000P	89/12/19	1:09:41	BLNK	1,593				
PR90000P	89/12/19	14:26:42	BLNK	1,539		RESTARTED		
FORM# TOTAL				3,132			2	
SMF006 - OUTPUT WRITER FORMS CONTROL REPORT							PAGE	2
SYSTEM 3081								
FORM JMTL								
JOBNAME	DATE	TIME	FORM NUMBER	RECORD COUNT	ROUTED	OPERATOR INTERVENTION	JOB COUNT	
OS10JEW	89/12/19	13:34:09	JMTL	135				
OS10JRN	89/12/20	10:31:19	JMTL	144				
OS10JRN	89/12/19	14:26:41	JMTL	225				
OS10JRW	89/12/20	14:26:43	JMTL	1,152				
FORM# TOTAL				1,656			4	
SYSID TOTAL				4,788			6	
FINAL TOTAL				4,788			6	

This sample lists output forms control information in the SMF data set.

## SMF Data Lost - SMF007

The SMF007 routine creates a report listing the date, time, elapsed time, and number of records lost in any interruption of SMF recording. The parameters that you pass allow for selection based on date and time.

### Syntax

```
%SMF007A starttime endtime startdate enddate [REPORT reportname]
```

```
%SMF007B [REPORT reportname]
```

starttime

Specify the time that the routine will begin reporting on lost SMF data. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on lost SMF data. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on lost SMF data. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on lost SMF data. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

## Operation

In order to include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF007.

### Input

```
%SMFILE
%SMF007A 130000 140000 890426 890426
%SMF007B
```

### Output

```
SMF007 - DATA LOST (SMF NOT RECORDED) REPORT
SYSTEM 3083
```

DATE	TIME	MINUTES ELAPSED TIME	NUMBER OF RECORDS LOST
89/04/26	13:51:37	.0483	11
89/04/26	13:51:38	.4166	47
89/04/26	13:51:40	.7183	19
SYSID TOTAL		1.1832	
FINAL TOTAL		1.1832	

This sample lists all occurrences of lost SMF data in the data set.

## Data Set Activity - SMF014

The SMF014 routine creates a report of data set activity including the accessing job name, date, and time. The parameters that you pass allow for selection, sorting, and totaling based on DATE, TIME, data set name (DSNAME), or JOBNAME.

## Syntax

```
%SMF014A selection selectstart selectend [REPORT reportname]
%SMF014B [REPORT reportname] [SORT sortflds [D]] +
[CONTROL cntrlflds [options]]
```

**selection**

Specify the field in the SMF record that determines the range for selection of records in this report. The specification of this parameter determines the valid values for the selectstart and selectend parameters. The value that you specify for selection must be one of the following:

**DATE** – Date determines selection. The selection range that selectstart and selectend specify must be dates in YYMMDD format. Valid values for selectstart and selectend are a six-byte alphanumeric field or actual six-digit numeric values enclosed in triple quotes.

**TIME** – Time determines selection. The selection range that selectstart and selectend specify must be times in HHMMSS format. Valid values for selectstart and selectend are a six-byte alphanumeric field or actual six-digit numeric values enclosed in triple quotes.

**DSNAME** – Data set name determines selection. The selection range that selectstart and selectend specify must be alphabetic literals representing data set names. Valid values for selectstart and selectend are a 44-byte alphanumeric field or a character string up to 44 characters long enclosed in triple quotes.

**JOBNAME** – Job name determines selection. The selection range that selectstart and selectend specify must be alphabetic literals representing job names. Valid values for selectstart and selectend are an eight-byte alphanumeric field or a character string up to eight bytes long enclosed in triple quotes.

**selectstart**

Specify the beginning of the selection range for the field chosen in the selection parameter. Valid values vary depending on the field chosen for the selection parameter.

**selectend**

Specify the end of the selection range for the field chosen in the selection parameter. Valid values vary depending on the field chosen for the selection parameter.

**[REPORT reportname]**

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

[SORT sortflds [D]]

Specify the field or fields that you want to use to sequence the report. Valid values are any of the field names described in the selection parameter. To specify multiple fields for sequencing, enclose the field names in single quotes and separate the field names by one blank space.

To specify a descending sort sequence, insert the letter D after the appropriate field name. You must use a blank space to separate the D from the previous and any subsequent field names. When you specify descending sequence, enclose all items in the parameter list in single quotes.

[CONTROL cntrlflds [options]]

Specify the field or fields that you want for control breaks in the reporting process. Valid values for cntrlflds are any of the field names described in the selection parameter. To specify multiple fields for control breaks, enclose the field names in single quotes and separate them by one blank space.

You can specify report processing options after the control break field. These options customize the report in relation to control break activities. These options are:

**NEWPAGE**—Causes a skip to top-of-page after processing is complete for the control break field.

**RENUM**—Performs the same function as NEWPAGE and also resets the page number to one following the control break.

**NOPRINT**—Suppresses printing of the summary line group for the control break that you specify.

If you specify any of these options, separate all items in the parameter list by one blank space and enclose the entire string in single quotes.

## Operation

To include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF014.

Input

```
%SMFILE
%SMF014A TIME ''080000'' ''090000''
%SMF014B SORT 'PROGRAM_ TIME' CONTROL PROGRAM
```

## Output

```

SMF014 - DATA SET ACTIVITY REPORT
        SYSTEM ID 3081
        SELECTED BY TIME
        SEQUENCED BY SYSID PROGRAM TIME

        DATA SET
JOBNAME          NAME          DATE          TIME          NUMBER OF
                                     DATASETS

LINK1            SYSTEM.LOADLIB      89/03/20      8:21:18
LINK1            SYSTEM.LOADLIB      89/03/20      8:21:19
PROGRAM TOTAL                                     2

NET              SYS1.VTAMLST         89/03/20      8:42:24
NET              SYS1.VTAMLST         89/03/20      8:42:53
NET              SYS1.VTAMLST         89/03/20      8:43:27
PROGRAM TOTAL                                     3

PAN1             SYSTEM.PANLIB        89/03/20      8:16:52
PROGRAM TOTAL                                     2

PAN2             SYSTEM.PANLIB        89/03/20      8:24:48
PAN2             SYSTEM.PANLIB        89/03/20      8:24:51
PAN2             SYSTEM.PANLIB        89/03/20      8:25:00
PROGRAM TOTAL                                     3

SMITH            SMITH.TESTFILE      89/03/20      8:49:16
SMITH            SMITH.TESTFILE      89/03/20      8:54:16
SMITH            SMITH.TESTFILE      89/03/20      8:57:36
PROGRAM TOTAL                                     3

TOPPER           SYSTEM.EZTVFM        89/03/20      8:25:35
TOPPER           SYSTEM.EZTVFM        89/03/20      8:25:35
PROGRAM TOTAL                                     2

TOPPER2          SYSTEM.EZTVFM        89/03/20      8:49:17
TOPPER2          SYSTEM.EZTVFM        89/03/20      8:49:17
PROGRAM TOTAL                                     2

TEST1            SYSTEM.TEST.LOAD     89/03/20      8:35:15
PROGRAM TOTAL                                     1

SYSID TOTAL                                     550

FINAL TOTAL                                         550

```

This sample lists all data set activity for all days present in the SMF data set. The report is sequenced by TIME and PROGRAM\_ with control breaks on the field PROGRAM\_ within system id (SYSID).

## Scratched Data Sets - SMF017

The SMF017 routine creates a report listing scratched data sets. A field that you specify within system ID (SYSID) sequences the report. The parameters that you pass allow for selection based on date and time.

### Syntax

```
%SMF017A starttime endtime startdate enddate [REPORT reportname]
```

```
%SMF017B [REPORT reportname] [SORT sortflds [D]]
```

starttime

Specify the time that the routine will begin reporting on scratched data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on scratched data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on scratched data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on scratched data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

[SORT sortflds [D]]

Specify the field or fields that you want to use to sequence the report. The value for sortfld must be one of the following:

- DATE
- TIME
- DSNNAME (data set name)
- JOBNAME
- USERID\_

If you want to specify multiple fields for sequencing, enclose the field names in single quotes and separate the field names by one blank space.

To specify a descending sort sequence, insert the letter D after the appropriate field name. You must use a blank space to separate the D from the previous and any subsequent field names. When you specify descending sequence, enclose all items in the parameter list in single quotes.

## Operation

In order to include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF017.

Input

```
%SMFILE
%SMF017A 000000 080000 891219 891219
%SMF017B SORT 'JOBNAME TIME'
```

Output

SMF017 - SCRATCHED DATA SET REPORT  
 SYSTEM 3081  
 SEQUENCED BY SYSID JOBNAME TIME

PAGE 1

DATA SET NAME	JOBNAME	DATE	TIME	USERID	NUMBER OF VOLUMES
SMITH.APT.R012ABS.T.APTMDLO	SMITH	89/12/19	7:40:01		1
SYS89353.T074258.RA000.SMITH.R0000003	SMITH	89/12/19	7:40:01		1
SYS89353.T074300.RA000.SMITH.R0000004	SMITH	89/12/19	7:40:01		1
SYS89353.T074301.RA000.SMITH.R0000005	SMITH	89/12/19	7:40:01		1
SYS89353.T074303.RA000.SMITH.R0000006	SMITH	89/12/19	7:40:01		1
SYS89353.T074304.RA000.SMITH.R0000007	SMITH	89/12/19	7:40:01		1
SYS89353.T074306.RA000.SMITH.R0000008	SMITH	89/12/19	7:40:01		1
SMITH.APT.R012ABS.T.APTMDLO	SMITH	89/12/19	7:40:01		1
SMITH.SPFLOG4.LIST	SMITH	89/12/19	7:40:01		1
SMITH.SPFLOG4.LIST	SMITH	89/12/19	7:40:01		1
SYS89353.T072234.RA000.RYAN.R0000001	RYAN	89/12/19	7:22:32		1
SYS89353.T072234.RA000.RYAN.R0000002	RYAN	89/12/19	7:22:32		1
SYS89353.T072234.RA000.RYAN.R0000003	RYAN	89/12/19	7:22:32		1
SYS89353.T072234.RA000.RYAN.R0000004	RYAN	89/12/19	7:22:32		1
SYS89353.T072235.RA000.RYAN2.R0000001	RYAN2	89/12/19	7:22:33		1
SYS89353.T072235.RA000.RYAN2.R0000002	RYAN2	89/12/19	7:22:33		1
SYS89353.T072235.RA000.RYAN2.R0000003	RYAN2	89/12/19	7:22:33		1
SYS89353.T072235.RA000.RYAN2.R0000004	RYAN2	89/12/19	7:22:33		1
RYAN2.SPFLOG1.LIST	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.EDIT001	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.EDIT001	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.EDIT001	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.EDIT001	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.BROWSE2	RYAN2	89/12/19	7:22:33		1
RYAN2.SPFLOG1.LIST	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.EDIT001	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.EDIT001	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.BROWSE2	RYAN2	89/12/19	7:22:33		1
RYAN2.SPFLOG1.LIST	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.BROWSE2	RYAN2	89/12/19	7:22:33		1
RYAN2.TESTDATA.ISPFOPFN.EDIT001	RYAN2	89/12/19	7:22:33		1
JACKSON.SPFTEMP0.CNTL	JACKSON	89/12/19	7:31:08		1
JACKSON.PANVALET.ISPFOPFN.EDIT002	JACKSON	89/12/19	7:31:08		1
SYS89353.T073108.RA000.JACKSON.R0000003	JACKSON	89/12/19	7:31:08		1
SYS89353.T073108.RA000.JACKSON.R0000004	JACKSON	89/12/19	7:31:08		1
SYS89353.T073527.RA000.JACKSON.R0000044	JACKSON	89/12/19	7:31:08		1
SYS89353.T073531.RA000.JACKSON.R0000045	JACKSON	89/12/19	7:31:08		1
SYS89353.T073108.RA000.JACKSON.R0000002	PEARSON	89/12/19	7:31:08		1
SYS89353.T073108.RA000.JACKSON.R0000001	JACKSON	89/12/19	7:31:08		1
SYS89353.T074026.RA000.JACKSONC.R0000001	JACKSONC	89/12/19	7:40:25		1
SYS89353.T074026.RA000.JACKSONC.R0000002	JACKSONC	89/12/19	7:40:25		1
SYS89353.T074112.RA000.JACKSONC.R0000001	JACKSONC	89/12/19	7:40:37		1
SYS89353.T074112.RA000.JACKSONC.R0000002	JACKSONC	89/12/19	7:40:37		1

This sample lists data sets scratched in the SMF data set. The report is sequenced by data set name, TIME, and JOBNAME within SYSID.

## Renamed Data Sets - SMF018

The SMF018 routine creates a report listing renamed data sets. The report is sequenced by an optionally specified field within system ID (SYSID). The parameters that you pass allow for selection based on date and time.

### Syntax

```
%SMF018A starttime endtime startdate enddate [REPORT reportname]
```

```
%SMF018B [REPORT reportname] [SORT sortflds [D]]
```

starttime

Specify the time that the routine will begin reporting on renamed data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on renamed data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on renamed data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on renamed data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

[SORT sortflds [D]]

Specify the field that you want to use to sequence the report. The value for sortfld must be one of the following:

- DATE
- TIME
- JOBNAME
- NEWNAME (new data set name)
- OLDNAME (old data set name)

To specify multiple fields for sequencing, enclose the field names in single quotes and separate the field names by one blank space.

To specify a descending sort sequence, insert the letter D after the appropriate field name. You must use a blank space to separate the D from the previous and any subsequent field names. When you specify descending sequence, enclose all items in the parameter list in single quotes.

## Operation

In order to include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF018.

Input

```
%SMFILE
%SMF018A 000000 240000 891219 891220
%SMF018B SORT 'JOBNAME TIME'
```

## Output

```

SMF018 - RENAMED DATA SET REPORT                                PAGE    1
SYSTEM 3081
SEQUENCED BY SYSID JOBNAME TIME

      OLD DATA SET          NEW DATA SET          VOL
      NAME                   NAME                   JOBNAME  DATE    TIME  USERID  CNT
PSISYS.MINIDISK.RETSVML.D195  PSISYS.OLD.MINIDISK.RETSVML.D195  DIRADREN 89/12/19 10:14:21 -    1
PSISYS.MINIDISK.SPARE03      PSISYS.MINIDISK.RETSVML.D195      DIRADREN 89/12/19 10:14:23 -    1
TECHS.OLD.MINIDISK.BROWWIN    TECHS.MINIDISK.CC255.YURKOVI      DIRADREN 89/12/20 16:46:07 -    1
JACKSON.P1032858.PANDD3      JACKSON.P1032858.PANDD3          JACKSON  89/12/19 10:50:16 -    1

```

This sample lists data sets renamed at any time of day in the SMF data set. The report is sequenced by TIME and JOBNAME within SYSID.

## Job Initiations - SMF020

The SMF020 routine creates a report listing jobs initiated by date, time, and programmer name. The report is sequenced by TIME, DATE, and JOBNAME within system ID (SYSID). The parameters that you pass allow for selection based on date and time.

### Syntax

```
%SMF020A starttime endtime startdate enddate [REPORT reportname]
```

```
%SMF020B [REPORT reportname]
```

starttime

Specify the time that the routine will begin reporting on initiated jobs. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on initiated jobs. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on initiated jobs. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on initiated jobs. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

## Operation

In order to include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF020.

Input

```
%SMFILE
%SMF020A 000000 070000 891219 891219
%SMF020B
```

Output

```

                                SMF020 - JOB INITIATION REPORT                                PAGE      1
                                SYSTEM 3081

```

JOBNAME	DATE	TIME	USERID	PROGRAMMER NAME	NUM OF JOBS
DBUSR054	89/12/19	0:01:26	-	DATA.CENTER BKUPS	1
DEALLOC	89/12/19	0:04:01			2
DEALLOC	89/12/19	0:18:47			
JOB	89/12/19	0:00:03			
JOB	89/12/19	0:01:24			2

This sample lists jobs initiated in the SMF data set.

## JES2 and JES3 Integrity - SMF049

The SMF049 routine creates a report listing occurrences of password invalidity and reason for failure of RJE station sign-on. The report is sequenced by TIME, DATE, and LINE within system ID (SYSID). The parameters that you pass allow for selection based on date and time.

### Syntax

```
%SMF049A starttime endtime startdate enddate [REPORT reportname]
```

```
%SMF049B [REPORT reportname]
```

starttime

Specify the time that the routine will begin reporting on JES2 and JES3 integrity. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on JES2 and JES3 integrity. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on JES2 and JES3 integrity. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on JES2 and JES3 integrity. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

## Operation

In order to include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you need to invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF049.

### Input

```
%SMFILE
%SMF049A 130000 140000 890426 890426
%SMF049B
```

### Output

```

                                SMF049 - JES INTEGRITY REPORT
                                SYSTEM 3083

KEY: TNF-TERM NOT DEFINED IP-INVALID PASSWORD LASO-LINE ALREADY SIGNED ON TASO-TERM ALREADY SIGNED ON

REMOTE      LINE      INVALID
            PASSWORD  DATE      TIME      REASON    MESSAGE
REMOTE*7    E*12    PASSWD34  89/04/26  13:51:37  TND       MESSAGE*7
REMOTE TOTAL

REMOTE*4    E*4E    PASSWD96  89/04/26  13:51:38  IP        MESSAGE*90
REMOTE*4    E*4E    PASSWD32  89/04/26  13:51:39  TND       MESSAGE*7
REMOTE TOTAL

SYSID TOTAL

FINAL TOTAL
```

This sample lists information concerning JES integrity in the SMF data set.

## VSAM Opens - SMF062

The SMF062 routine creates a report listing each VSAM data set opened and each VSAM open attempt that failed due to an invalid password. The report is sequenced by TIME, DATE, and CLUSTER (VSAM data set name) within system ID (SYSID). The parameters that you pass allow for selection based on date and time.

## Syntax

%SMF062A starttime endtime startdate enddate [REPORT reportname]

%SMF062B [REPORT reportname]

starttime

Specify the time that the routine will begin reporting VSAM data set open activity. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on VSAM data set open activity. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on VSAM data set open activity. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on VSAM data set open activity. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

## Operation

To include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you must invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.



starttime

Specify the time that the routine will begin reporting on deleted VSAM entries. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on deleted VSAM entries. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on deleted VSAM entries. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on deleted VSAM entries. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

## Operation

To include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you must invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF067.

Input

```
%SMFILE
%SMF067A 000000 130000 890319 890322
%SMF067B
```

## Output

SMF067 - DELETED VSAM ENTRIES REPORT  
SYSTEM 3083

ENTRY CATALOG	JOBNAME	DATE	TIME	FUNCTION COMPLETED	STRUCTURE
SYS.PANLINK.BOX1 USERCAT.XUSER1B	JONESPL	89/03/20	11:58:01	UNCATALOGED	VSAM CLUSTER
SYS.PANLINK.BOX1 USERCAT.XUSER1B	JONESPL	89/03/20	12:07:39	UNCATALOGED	VSAM CLUSTER
SYS.PANLINK.BOX1.DATA USERCAT.XUSER1B	JONESPL	89/03/20	12:07:39	UNCATALOGED	VSAM DATA
SYS.PANLINK.BOX1.DATA USERCAT.XUSER1B	JONESPL	89/03/20	11:58:01	UNCATALOGED	VSAM DATA
SYS.PANLINK.BOX2 USERCAT.XUSER1B	JONESPL	89/03/20	12:03:31	UNCATALOGED	VSAM CLUSTER
SYS.PANLINK.BOX2 USERCAT.XUSER1B	JONESPL	89/03/20	12:12:20	UNCATALOGED	VSAM CLUSTER
SYS.PANLINK.BOX2.DATA USERCAT.XUSER1B	JONESPL	89/03/20	12:12:20	UNCATALOGED	VSAM DATA
SYS.PANLINK.BOX2.DATA USERCAT.XUSER1B	JONESPL	89/03/20	12:03:31	UNCATALOGED	VSAM DATA
SYS.PANLINK.TEST USERCAT.XUSER1B	JONESPL	89/03/20	12:14:06	UNCATALOGED	VSAM CLUSTER
SYS.CATLIBU USERCAT.XUSER1B	JOHNSON	89/03/20	12:36:59	UNCATALOGED	VSAM CLUSTER
SYS.CATLIBU.DATA USERCAT.XUSER1B	JOHNSON	89/03/02	12:36:59	UNCATALOGED	VSAM DATA
SYS.CATLIBU.INDEX USERCAT.XUSER1B	JOHNSON	89/03/20	12:37:01	UNCATALOGED	VSAM INDEX
VSAMDSET.TFC7D476.DFD85078.T98DAFEA.TFC7D476 USERCAT.XUSER1B	JONESPL	89/03/20	12:14:06	UNCATALOGED	VSAM DATA
SYSID TOTAL					
FINAL TOTAL					

This example lists VSAM entries deleted in the SMF data set.

## VSAM Renames - SMF068

The SMF068 routine creates a report listing renamed VSAM data sets. The report indicates both the old and new data set name. The report is sequenced by old data set name within system ID (SYSID). The parameters that you pass allow for selection based on date and time.

### Syntax

```
%SMF068A starttime endtime startdate enddate [REPORT reportname]
```

```
%SMF068B [REPORT reportname]
```

starttime

Specify the time that the routine will begin reporting on renamed VSAM data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

endtime

Specify the ending time for reporting on renamed VSAM data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in HHMMSS format based on a 24-hour clock.

startdate

Specify the date that the routine will begin reporting on renamed VSAM data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

enddate

Specify the ending date for reporting on renamed VSAM data sets. A valid value is an actual numeric value or the name of a field containing a numeric value. Express values in YYMMDD format.

[REPORT reportname]

For the second and all subsequent invocations of the same SMF routine, specify a unique reportname that associates the processing performed in the A routine with the report of the B routine. For each reportname specified on an A routine, the identical reportname must be specified on a B routine of the same SMF report type. For the first occurrence of an SMF routine, the routine supplies a default name and does not require the parameter.

## Operation

To include the required file and field definitions, all SMF audit routines require you to invoke the SMFILE routine prior to the audit routine. If you invoke multiple SMF routines, you must invoke SMFILE only once, followed by the invocation of the desired SMF routines. For details, see [SMF Audit Routines](#) earlier in this chapter.

## Example

The following is an example of SMF068.

### Input

```
%SMFILE
%SMF068A 130000 140000 890426 890426
%SMF068B
```

### Output

```

                                SMF068 - RENAMED VSAM ENTRIES REPORT
                                SYSTEM 3083
                                OLD DS NAME
                                CATALOG
                                NEW DS NAME
                                JOBNAME
                                DATE
                                TIME
JONES.VSAM.FILE
USERCAT.XUSER1B
                                JONES.SAVE.FILE
                                JONES
                                89/04/26
                                13:51:38
SAMPLE.FILE
USERCAT.XUSER1B
                                SAMPLE.VSAM.FILE
                                JOHNSON
                                89/04/26
                                13:51:39
SMITH.TEST1
USERCAT.XUSER1B
                                SMITH.PROD
                                SMITHVS
                                89/04/26
                                13:51:37
SYSID TOTAL
FINAL TOTAL
```

This sample lists VSAM data sets renamed in the SMF data set.

## Frequency Distribution of SMF Records - SMFCNT

The SMFCNT routine creates a frequency distribution of all SMF record types on the input SMF file. No selection criteria is available.

### Syntax

```
%SMFCNT
```

## Operation

You can use the SMFCNT routine before executing any of the other SMF audit routines. The SMFCNT routine provides a frequency distribution of all SMF record types. If there are no records of a specific type present on the SMF data set, then the report does not list a frequency distribution for that record type.

The SMFCNT routine contains the file definitions from SMFILE. Therefore, you must not invoke the SMFILE routine prior to SMFCNT. Also, you cannot use SMFCNT with the other SMF reporting routines.

## Example

The following is an example of SMFCNT.

Input

```
%SMFCNT
```

Output

```

                                FREQUENCY DISTRIBUTION OF TYPE
                                INPUT FILENAME SMFILE
                                PAGE      1

TYPE      COUNT      PCT
  2         1         .0
  3         1         .0
  4       1,439       6.3   *****
  5         466       2.0   **
  6          82         .4
  9          4         .0
 10         3         .0
 11         7         .0
 14       5,695      25.0   *****
 15       3,795      16.7   *****
 17       2,070       9.1   *****
 18          4         .0
 19         64         .3
 20        505       2.2   **
 21        185         .8   *
 22          1         .0
 26         905       4.0   ****
 30       2,484      10.9   *****
 32         36         .2
 34         37         .2
 35         37         .2
 40       2,206       9.7   *****
 50         25         .1
 57         271       1.2   *
 60         476       2.1   **
 61          3         .0
 62         682       3.0   ***
 63          3         .0
 64       1,266       5.6   *****
 65          5         .0
 66          3         .0
 90          1         .0
22,762    100.0

```

This sample lists the frequency of occurrence of the various types of SMF records contained on the input SMF file.

## SMF Record Field Definitions

The SMF record field definition macros provide field definitions for the majority of SMF record types. A macro has been created for most SMF record types that contain field definitions that follow the naming conventions listed in the IBM guide *OS/390 MVS System Management Facilities (SMF)*. With the release of MVS/XA 2.2.0 and MVS/ESA 3.1.3, some SMF records changed. To reflect these changes, three sets of Toolkit macros define the SMF records. The functionality of these macros is identical to their predecessors; they define the MVS/XA 2.2.0 or MVS/ESA 3.1.3 SMF record layouts.

You invoke these macros by coding a percentage sign (%), then the appropriate macro name. The following is a list of the macro names, the SMF record types, and a description of the SMF record types. The macros listed in column -A- are to be used with SMF records generated by MVS systems prior to MVS/XA 2.2.0 or MVS/ESA 3.1.3. Those listed in column -B- are for SMF records produced by MVS/XA 2.2.0. Those listed in column -C- are for SMF records produced by MVS/ESA 3.1.3.

Macro Name			SMF Record	SMF Record Description
-A-	-B-	-C-	Type	
SMFILE	S22FILE	SE13FILE	-	SMF Data File Definitions for Audit Routines
SMFR00	S22R00	SE13R00	0	IPL
SMFR02	S22R02	SE13R02	2	Dump Header
SMFR03	S22R03	SE13R03	3	Dump Trailer
SMFR04	S22R04	SE13R04	4	Step Termination
SMFR05	S22R05	SE13R05	5	Job Termination
SMFR06J2	S22R06J2	SE13R06A	6	JES2 Output Writer
SMFR06J3	S22R06J3	SE13R06B	6	JES3 Output Writer
SMFR07	S22R07	SE13R07	7	Data Lost
SMFR08	S22R08	SE13R08	8	I/O Configuration
SMFR09	S22R09	SE13R09	9	VARY ONLINE
SMFR10	S22R10	SE13R10	10	Allocation Recovery
SMFR11	S22R11	SE13R11	11	VARY OFFLINE
SMFR14	S22R14	SE13R14	14	INPUT or RDBACK Data Set Activity
SMFR15	S22R15	SE13R15	15	OUTPUT, UPDAT, INOUT or OUTIN Data Set Activity
SMFR17	S22R17	SE13R17	17	Scratch Data Set Status
SMFR18	S22R18	SE13R18	18	Rename Data Set Status
SMFR19	S22R19	SE13R19	19	Direct Access Volume
SMFR20	S22R20	SE13R20	20	Job Initiation
SMFR21	S22R21	SE13R21	21	Error Statistics by Volume
SMFR22	S22R22	SE13R22	22	Configuration
SMFR23	S22R23	SE13R23	23	SMF Status Record
SMFR25	S22R25	SE13R25	25	JES3 Device Allocation
SMFR26J2	S22R26J2	SE13R26A	26	JES2 Job Purge
SMFR26J3	S22R26J3	SE13R26B	26	JES3 Job Purge
SMFR30	S22R30	SE13R30	30	Common Address Work Record
SMFR31	S22R31	SE13R31	31	TIOC Initialization
SMFR32	S22R32	SE13R32	32	TSO User Work Accounting Record
SMFR34	S22R34	SE13R34	34	TS-Step Termination

SMFR35	S22R35	SE13R35	35	LOGOFF
SMFR40	S22R40	SE13R40	40	Dynamic DD
SMFR43J2	S22R43J2	SE13R43A	43	JES2 Start
SMFR43J3	S22R43J3	SE13R43B	43	JES3 Start
SMFR45J2	S22R45J2	SE13R45A	45	JES2 Withdrawl
SMFR45J3	S22R45J3	SE13R45B	45	JES3 Stop
SMFR47J2	S22R47J2	SE13R47A	47	JES2 SIGNON/Start Line (BSC only)
SMFR47J3	S22R47J3	SE13R47B	47	JES3 SIGNON/Start Line LOGON
SMFR48J2	S22R48J2	SE13R48A	48	JES2 SIGNOFF/Stop Line (BSC only)
SMFR48J3	S22R48J3	SE13R48B	48	JES3 SIGNOFF/Stop Line LOGON
SMFR49J2	S22R49J2	SE13R49A	49	JES2 Integrity (BSC only)
SMFR49J3	S22R49J3	SE13R49B	49	JES3 Integrity
SMFR50	S22R50	SE13R50	50	ACF/VTAM Tuning Statistics
SMFR52	S22R52	SE13R52	52	JES2 LOGON/Start Line (SNA only)
SMFR53	S22R53	SE13R53	53	JES2 LOGOFF/Stop Line (SNA only)
SMFR54	S22R54	SE13R54	54	JES2 Integrity (SNA only)
SMFR55	S22R55	SE13R55	55	JES2 Network SIGNON Record
SMFR56	S22R56	SE13R56	56	JES2 Network Integrity Record
SMFR57J2	S22R57J2	SE13R57A	57	JES2 Network SYSOUT Transmission Record
SMFR57J3	S22R57J3	SE13R57B	57	JES3 Network SYSOUT Transmission Record
SMFR58	S22R58	SE13R58	58	JES2 Network SIGNOFF Record
SMFR62	S22R62	SE13R62	62	VSAM Component or Cluster Opened
SMFR63	S22R63	SE13R63	63	VSAM Entry Deleted
SMFR64	S22R64	SE13R64	64	VSAM Component or Cluster Status
SMFR67	S22R67	SE13R67	67	VSAM Entry Deleted
SMFR68	S22R68	SE13R68	68	VSAM Entry Renamed
SMFR69	S22R69	SE13R69	69	VSAM Data Space Defined, Extended or Deleted
SMFR70	S22R70	SE13R70	70	CPU Activity
SMFR71	S22R71	SE13R71	71	Paging Activity
SMFR72	S22R72	SE13R72	72	Workload Activity
SMFR73	S22R73	SE13R73	73	Channel Activity
SMFR74	S22R74	SE13R74	74	Device Activity
SMFR75	S22R75	SE13R75	75	Page/Swap Data Set Activity
SMFR76	S22R76	SE13R76	76	Trace Activity
SMFR77	S22R77	SE13R77	77	Enqueue Activity
SMFR79	S22R79	SE13R79	79	Monitor II Activity
SMFR82	S22R82	SE13R82	82	Security
SMFR90	S22R82	SE13R90	90	System Status Record

## Operation

These macros contain the field definitions for the SMF record types. The use of these macros with CA-Easytrieve Plus logic eliminates the time-consuming task of defining the fields that the SMF records contain.

These macros do not contain a CA-Easytrieve Plus FILE statement. You must supply the FILE statement and all associated processing logic. The following example invokes SMF field definition macros:

```
FILE SMFFILE
%SMFR20
%SMFR05
%SMFR04
SMFTYPE  2 1 B
JOB INPUT SMFFILE
IF SMFTYPE NE 20, 4, 5
  GO TO JOB
END-IF
.
.
user-defined processing
.
.
```

In this example, the file SMFFILE contains the record definitions for Job Initiation (SMFR20), Job Termination (SMFR05), and Step Termination (SMFR04). A common field name, SMFTYPE, provides a facility to screen the input file to accept only record types 20, 5, and 4. The IF statement screens the input file by testing the SMFTYPE field. If the record type is not equal to 20, 5, or 4, the GO TO JOB statement transfers control to the JOB statement and effectively bypasses the user-defined processing for that record.

Fields defined in the SMF record field definition macros follow the exact layout as the IBM SMF Guide defines, with the following exceptions:

- Many SMF records contain variable-length sections. These sections usually contain a variable number of fixed-length subsections. A field within the record contains the number of subsections which comprise a variable length section. To provide a method for accessing this data with CA-Easytrieve Plus, the SMF record field definition macro uses an indexing technique. Consult the CA-Easytrieve Plus *Reference Guide* for details regarding the use of indexing.
- Several fields in SMF records are defined with data structures not currently supported by CA-Easytrieve Plus. These include:
  - Variable length fields
  - Fields greater than 254 bytes
  - Binary fields greater than four bytes in length

The SMF record field definition macro uses different methods to provide definitions for these nonsupported data types depending on the use of the information. For information regarding the methods used, see the IBM SMF Guide and the appropriate SMF record field definition macro.



# Advanced SMF Reporting Facility (JIF)

---

The OS Job Information Facility (JIF) is a system for reporting on records obtained from IBM System Management Facilities (SMF).

Processing SMF-generated data for use in statistical analysis, cost accounting, and customer billing may very easily become an application nightmare. JIF retrieves SMF records, consolidates them, creates files of SMF data, and produces reports on this data without the need for you to develop sophisticated application software to interface with SMF.

For most effective use of JIF, you must have knowledge of SMF records and know the SMF parameters in effect at your installation.

## JIF Capabilities

JIF lets you:

- Report on SMF record types 00, 04, 05, 06, 07, 20, 26, and 40. Optionally, report on record types 34 and 35, or type 30.
- Report on other SMF record types through the user exit facility.
- Consolidate the SMF data into job and, optionally, TSO session representations.
- Create an SMF data file tailored to your needs.
- Produce preformatted statistical reports, using the supplied routines.
- Create customized reporting routines of your own, with CA-Easytrieve Plus.
- Receive audit reports on your use of the JIF system.
- Report on SMF records generated before MVS/XA 2.2.0, or SMF records generated by MVS/XA 2.2.0 or MVS/ESA 3.1.3.

## Facility Description

JIF has five components:

- JIFOPTS Options Module
- JIFSEL SMF Data Processor Function
- A User Exit Facility
- A CA-Easytrieve Plus Read Input Exit (JIFRDREX)
- Statistical Reporting Routines

### JIFOPTS

The options module, JIFOPTS, provides information to JIFSEL indicating which of the SMF record types you want processed and the content of the consolidated record file to be produced. This provides a degree of customization in the consolidated file.

### JIFSEL

JIFSEL processes the data produced by SMF, consolidating all SMF records for a job or TSO session into a single record. This record is then written to the consolidated file. JIFSEL selects both automatically and on the basis of the options you specify in JIFOPTS.

### User Exit Facility

The user exit facility gives you the ability to further customize the consolidated file produced by JIFSEL. Use the EXIT1 routine to select and the EXIT2 routine to process any additional SMF record types you want to report on.

### Read Input Exit

JIFRDREX reads the consolidated file, then formats and presents a fixed-length record to Toolkit. The JIF routines can be used to generate reports on this file.

### Statistical Reporting Routines

Toolkit routines are provided, which allow you to produce a variety of statistical reports from the data in the consolidated file. You can create additional customized reporting routines, using CA-Easytrieve Plus.

## Facility Operation

Execution of JIF is a two-step process:

1. Execute JIFSEL, which creates the consolidated file.
2. Read the consolidated file and produce a report by invoking a Toolkit routine.

The following JCL illustrates this process. The first step executes JIFSEL to create the consolidated file and audit file from the SMF data. The second executes CA-Easytrieve Plus and invokes a Toolkit routine, OSJIF03.

```
//jobname JOB accounting.info
//STEP1 EXEC PGM=JIFSEL
//STEPLIB DD ...
//SYSPRINT DD SYSOUT=A
//SYSUDUMP DD SYSOUT=A
//SYSOUT DD SYSOUT=A
//PJPRINT DD SYSOUT=A
//PJSORTIN DD DSN=user.SMF.dataset,DISP=(OLD,KEEP),UNIT=TAPE,
// VOL=SER=xxxxxx
//SORTOUT DD DSN=JIF.consol.idated.file,DISP=(NEW,CATLG,DELETE),
// SPACE=(CYL,(10,10),RLSE),UNIT=SYSDA,VOL=SER=xxxxxx
//PJAUDIT DD DSN=JIF.audit.file,DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(1,1),RLSE),UNIT=SYSDA,VOL=SER=xxxxxx
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,5)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,5)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,5)
/*
//STEP2 EXEC PGM=EZTPA00
//STEPLIB DD ...
//PANDD DD DSN=PAPL.macro.library,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,5)
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,5)
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,5)
//CONSOL DD DSN=JIF.consol.idated.file,DISP=SHR
//AUDIT DD DSN=JIF.audit.file,DISP=SHR
//EZTVFM DD UNIT=SYSDA,SPACE=(4096,(100,100))
//SYSIN DD *
%JIFREC YNNNNNNN
%OSJIF03 84003 84104
/*
//
```

### MVS/XA 2.2.0 Users

For users wanting to use MVS/XA 2.2.0 SMF records, append ,PARM='MVS(220)' to the line that starts with //STEP1, and verify that the SMF record input file contains only records from that release of MVS SMF. The default is pre-MVS/XA 2.2.0 SMF records so that existing operational job streams will run without modification.

Example:

```
//STEP1      EXEC PGM=JIFSEL , PARM='MVS(220)'
```

Pre-MVS/XA 2.2.0 SMF or MVS/XA 2.2.0 SMF and later routines can both be used on a processor running pre-2.2.0, 2.2.0, or post-2.2.0 releases of MVS as long as the appropriate data for the level of JIF/SMF routines selected is used. Both sets of routines can be run on the same processor, given that the preceding parameter is appended to the JCL.

## MVS/ESA 3.1.3 Users

For users wanting to use MVS/ESA 3.1.3 SMF records, append ,PARM='MVS(313)', to the line that starts with //STEP1, and verify that the SMF record input file contains only records from that release of MVS SMF. The default is pre-MVS/ESA 3.1.3 SMF records so that existing operational job streams will run without modification.

Example:

```
//STEP1      EXEC PGM=JIFSEL , PARM='MVS(313)'
```

Pre-MVS/ESA 3.1.3 SMF or MVS/ESA 3.1.3 SMF and later routines can both be used on a processor running pre-3.1.3, 3.1.3, or post-3.1.3 releases of MVS as long as the appropriate data for the level of JIF/SMF routines selected is used. Both sets of routines can be run on the same processor, given that the preceding parameter is appended to the JCL.

## JIFOPTS

Certain JIFSEL features are optional. The load module that specifies all options is JIFOPTS.

At installation, a model JIFOPTS module that contains all defaults is established. If your environment requires options other than those JIFOPTS supplies, you must link edit a new JIFOPTS.

For details on how to relink JIFOPTS, and for a description of the default and optional values, see the [Installation Guide](#).

---

## JIFSEL

JIFSEL has three functional phases during its execution: record selection, sorting, and data consolidation.

**SMF Record Selection** – During the record selection phase, JIFSEL loads JIFOPTS to determine which SMF record types are to be selected. JIF processes certain SMF record types based on defaults. Others are selected by your exit routines.

Each time an SMF record is read, JIFSEL determines if the record is selected for processing. If selected, JIFSEL builds a sort key from the record. The record is then passed to your operating system sort. If not selected, the program defined by the EXIT1 parameter of the JIF options table is invoked.

Depending on the action taken by your exit, the SMF record is either bypassed from further processing or forced into the processing stream. When a record is forced, JIFSEL builds the sort key, and the record is passed on to your sort.

**Sorting by Job or TSO Session** – JIFSEL builds a 28-byte sort key from information in the SMF record. This is done for all records processed. The actual structure of the JIF sort key is discussed in the *CA-Easytrieve Plus Toolkit [Installation Guide](#)*. You must consider the structure of the sort key when forcing some types of SMF records. The SMF records selected by JIFSEL or by your exit are sorted into a chronological order by job or TSO session.

**Consolidating SMF Records** – After the sort, JIFSEL collects information from multiple SMF records into a single consolidated record. (See [Consolidated Record Fields](#) later in this chapter.)

During this phase of operation, JIF communicates with the program defined by the EXIT2 parameter in JIFOPTS in order to process records selected by your EXIT1 routine. If EXIT2 is not specified, JIFSEL bypasses any records that are forced by EXIT1 when writing the consolidated record.

### SMF Record Selection

JIFSEL selects SMF records in three ways:

- By default
- Through parameters specified in the options macro
- By your exit routines

The record types selected by each of these techniques are shown in the following table:

JIFSEL Default	Options Macro	User Exit Facility
00, 04, 05	30 (subtypes 01, 04, 05)	All other SMF record types
06, 07, 20	01, 04, 05)	
26, 40	Batch and TSO; 34, 35 (TSO)	

The left-hand column lists the SMF record types processed by JIFSEL.

Column two indicates additional record types you can process automatically by modifying parameters in the options module. For example,

- Type-30 records can be processed instead of types 20, 04, 40, and 05
- TSO data (types 34 and 35) can be processed

Column three indicates the SMF record types you can process by means of your exit routines.

## Batch Environment

JIFSEL processes eight SMF record types from the batch environment:

Type 00 - IPL  
Type 04 - Step Termination (batch job)  
Type 05 - Job Termination (batch job)  
Type 06 - JES2 or JES3 Output Writer  
Type 07 - Lost Data  
Type 20 - Job Initiation (batch job)  
Type 26 - JES2 or JES3 Job Purge  
Type 40 - Dynamic DD.

Optionally, you can process type-30 (common address space work area) records. For details on the JIF options table, see the [Installation Guide](#).

## TSO Environment

If the option TSO=YES is in effect, JIFSEL processes the following SMF record types in addition to those listed on the previous page:

Type 20 - TSO Job Initiation  
Type 34 - TSO Step Termination  
Type 35 - Logoff.

Type-30 records can also be processed for TSO environments. Set the options macro parameter SMF30=YES and concurrently set TSO=YES.

## SMF Record Content

The SMF data in the consolidated file, as a result of JIFSEL execution, is derived from the SMF record sources described in the following table.

Definitions for and descriptions of these records can be found in the appropriate IBM guides on *System Management Facilities (SMF)*.

SMF Record Type	Record Name	Record Contents
20	Job initiation	Job name System identification User identification Programmer's name Accounting information (first 24 positions)
04		Step Termination Step name Program name Job name System identification Step start date and time Step termination date and time Step completion code Storage allocation Storage used Step CPU time (for OS/390 and above this represents SRB + TCB times) Step CPU time under SRB (OS/390 and above only) Step CPU time under TCB Device counts; EXCP counts for devices Page-ins and page-outs Number of address space swap sequences Number of VIO page-ins and page-outs Number of service units Residence times Number of page seconds

SMF Record Type	Record Name	Record Contents
05	Job termination	Job name System identification Job termination time and date Job start time and date Number of card images read by job Job priority Resident time Job input class Storage protect key Job CPU time (for OS/390 and above this represents SRB + TCB times) Job CPU time under SRB (OS/390 and above only) Job CPU time under TCB Job transaction active time Performance group number of last step
06	Output Writer	Time and date output was completed System identification Job name Sysout class Writer start date and time Number of logical records written Form number Approximate page count Logical device name
07	Data lost	Number of type-07 records processed during driver execution. For each 07 record processed: <ul style="list-style-type: none"> <li>■ Date and time record loss began</li> <li>■ Date and time records stopped</li> <li>■ Number of records lost</li> </ul>
40	Dynamic D.D.	Information is equivalent to the data set parts of type-04 records (device counts and EXCP counts for the devices)
30	Common Address Space Work Area	Information is the same as the record types (20, 04, 05 and 40) it replaces). SMF type-30, subtypes 1, 4, 5 are processed in place of SMF types 20, 04, 05, and 40.
34	TSO Step termination	Essentially the same information as type-04 records. Two additional fields are: <ul style="list-style-type: none"> <li>■ Number of lines of terminal output (number of TPUTS issued)</li> <li>■ Number of lines of terminal input (number of TGETS satisfied)</li> </ul>

SMF Record Type	Record Name	Record Contents
35	Logoff	Essentially the same information as type-05 records. Two additional fields are: <ul style="list-style-type: none"> <li>■ Number of lines of terminal output (number of TPUTS issued)</li> <li>■ Number of lines of terminal input (number of TGETS satisfied)</li> </ul>
26	Job Purge	System identification Job name Job number Job class Job priority Number of input statements for job

## JIFSEL Data Sets

JIFSEL creates three output data sets: the consolidated record file, the audit file, and the audit report.

**Consolidated Record File**— The JIF consolidated file contains one record for each job. If requested, each record can contain individual step and spool information.

Each record contains a 256-position user portion for data you include through exit processing.

**Audit File**— A record is created for the audit file each time JIFSEL is executed. The record contains input data set name, number of records read, Initial Program Load (IPL) records (SMF type-00) read, data lost (SMF type-07 records), and number of records processed.

**Audit Report**— The JIF audit report is a report on the data recorded and stored in the JIF audit file.

## Consolidated Record Fields

The following pages provide you with the field names used by the JIF routines when accessing the consolidated file. The file and field definitions are contained in the macro JIFREC. When using these fields, see the parameters of the JIFOPTS options module, as the contents of certain fields are developed based on parameters specified in JIFOPTS.

The description that follows describes the record presented to the JIF routines.

The area beginning at position 642 in the record, labeled Scratch Pad Area, is used by the routines to store additional information in the records prior to sorting.

```

FILE CONSOL  EXIT (JIFDREX USING ('&FLAGS'))  WORKAREA (1024)
*****
*
*          COMPUTER ASSOCIATES INCORPORATED          *
*
*          OS JOB INFORMATION FACILITY VERSION 1.0    *
*
*  THE FOLLOWING MACRO DESCRIBES THE INPUT RECORD    *
*  AS SEEN BY EASYTRIEVE. THE FIRST EIGHT BYTES    *
*  ARE AVAILABILITY FLAGS INDICATING WHEN PORTIONS  *
*  OF THE RECORD ARE PRESENT.                       *
*  XFLAG1 = Y WHEN JOB PORTION PRESENT.             *
*  XFLAG2 = Y WHEN STEP PORTION PRESENT.            *
*  XFLAG3 = Y WHEN SPOOL PORTION PRESENT.          *
*  XFLAG4 THROUGH XFLAG7 RESERVED.                 *
*  XFLAG8 = Y WHEN USER PORTION PRESENT.           *
*
*****
XFLAG1  1  1  A . XFLAG2  2  1  A . XFLAG3  3  1  A .
XFLAG4  4  1  A . XFLAG5  5  1  A . XFLAG6  6  1  A .
XFLAG7  7  1  A . XFLAG8  8  1  A .
*****
*
*  THIS SECTION OF THE MACRO DESCRIBES THE JOB PORTION. *
*
*****
CMSOURCE      9      1  A
CMFLAG1      10     1  A
CMFLAG2      11     1  A
CMFLAG3      12     1  A
CMFLAG4      13     1  B
CMFLAG5      14     1  B
CMFLAG6      15     1  B
CMFLAG7      16     1  B
CMTOTSTP     17     2  B  0
CMOFFSTP     19     2  B
CMTOTSPL     21     2  B  0
CMOFFSPL     23     2  B
CMCOUNT1    25     4  B
CMCOUNT2    29     4  P
CMSYSID      33     4  A
CMJOBNM      37     8  A      HEADING ('JOB' 'NAME')
CMJOBNO      45     2  U      HEADING ('JOB' 'NO.')
CMJOBPT      47     2  U      HEADING ('P' 'T' 'Y')
CMJOBCL      49     1  A      HEADING ('C' 'L')
CMPRKEY      50     1  U
CMPRGNAM     51     20  A      HEADING ('PROGRAMMER' 'NAME')
CMPRGNSH     51     14  A
CMUSRID      71     8  A      HEADING ('USER' 'INFORMATION')
CMACCT       79     24  A
CMABEND      103    1  A
CMTSTT       104    3  U      MASK ('99/99/99')
CMTMSTT      107    3  U      MASK ('Z9:99:99') +
                                HEADING ('START' 'TIME')
CMTMSTP      110    3  U      MASK ('Z9:99:99') +
                                HEADING ('STOP' 'TIME')
CMTMELAP     113    5  U  4    HEADING ('ELAPSED' 'TIME')
CMTMCPU      118    4  U  4    HEADING ('CPU' 'TIME')
CMTMTCB      122    4  U  4    HEADING ('TCB' 'TIME')
CMTMSRB      126    4  U  4    HEADING ('SRB' 'TIME')
CMTMACT      130    5  U  4    HEADING ('ACTIVE' 'TIME')
CMTMRES      135    5  U  4    HEADING ('RESIDENT' 'TIME')
CMTMALD      140    2  U  2    HEADING ('ALLOCATION' 'TIME')
CMTMRDR      142    4  U  4    HEADING ('RDR' 'TIME')
CMTMWRQ      146    4  U  4    HEADING ('WRITER' 'TIME')
CMTMWTR      150    4  U  4

```

CMTMTURN	154	5	U	4	HEADING ('TURNAROUND' 'TIME')
CMCARDR	159	3	U	0	
CMCARDP	162	3	U	0	
CMLINES	165	4	U	0	HEADING ('LINES' 'PRINTED')
CMTAPE	169	1	U	0	
CMDISK1	170	1	U	0	
CMDISK2	171	1	U	0	
CMDISKX	172	1	U	0	
CMOTHER	173	1	U	0	
CMIOTP	174	3	U	0	HEADING ('TAPE I/O' 'COUNT')
CMIODK1	177	3	U	0	
CMIODK2	180	3	U	0	
CMIODKX	183	3	U	0	
CMIOOTH	186	3	U	0	
CMVIOIN	189	3	U	0	
CMVIOOT	192	3	U	0	
CMSWAPIN	195	3	U	0	
CMSWAPOT	198	3	U	0	
CMPAGEIN	201	3	U	0	HEADING ('PAGEIN' 'COUNT')
CMPAGEOT	204	3	U	0	HEADING ('PAGEOUT' 'COUNT')
CMADDSP	207	2	U	0	HEADING ('SWAP' 'COUNT')
CMPGSEC	209	4	U	0	HEADING ('PAGE' 'SECONDS')
CMSECV	213	4	U	0	HEADING ('SERVICE' 'UNITS')
CMPERFGP	217	2	U		HEADING ('PER.' 'GROUP')
CMCOREAL	219	4	B	0	
CMCOREUS	223	2	B	0	
CMTSOTG	225	4	U	0	HEADING ('TSO' 'GETS')
CMTSOTP	229	4	U	0	HEADING ('TSO' 'PUTS')
DATE	104	1	U		
STRTTM	107	1	U		
CNTTAPE	169	1	A		

\*\*\*\*\*  
 \* THIS SECTION DESCRIBES THE JOB STEP PORTION. \*  
 \* \*  
 \*\*\*\*\*

CSNAME	233	8	A		HEADING ('STEP' 'NAME')
CSPROGN	241	8	A		HEADING ('PROGRAM' 'NAME')
CSCOREAL	249	4	B	0	
CSCOREUS	253	2	B	0	
CSABND	255	4	A		HEADING ('ABEND' 'CODE')
CSSTDT	259	3	U		MASK ('99/99/99') + HEADING ('STEP' 'RUN' 'DATE')
CSSTTM	262	3	U		MASK ('99:99:99') + HEADING ('STEP' 'START' 'TIME')
CSSPTM	265	3	U		MASK ('99:99:99')
CSELAPT	268	4	U	4	HEADING ('STEP' 'ELAPSED' 'TIME')
CSCPU	272	3	U	4	HEADING ('STEP' 'CPU' 'TIME')
CSTCB	275	3	U	4	
CSSRB	278	3	U	4	
CSACT	281	4	U	4	
CSRES	285	4	U	4	
CSALDEL	289	2	U	2	
CSTAPNUM	291	1	U	0	
CSDISK1	292	1	U	0	
CSDISK2	293	1	U	0	
CSDISKX	294	1	U	0	
CSOTHNM	295	1	U	0	
CSIOTP	296	3	U	0	
CSIODK1	299	3	U	0	
CSIODK2	302	3	U	0	
CSIODKX	305	3	U	0	
CSIOOTH	308	3	U	0	
CSVIOIN	311	3	U	0	
CSVIOOT	314	3	U	0	
CSSWAPIN	315	3	U	0	

```
CSSWAPOT 320 3 U 0
CSPAGEIN 323 3 U 0
CSPAGEOT 326 3 U 0
CSADDSP 329 2 U 0
CSPGSEC 331 4 U 0
CSSERV 335 4 U 0
CSTSOTG 339 4 U 0
CSTSOTP 343 4 U 0
*****
*
* THIS SECTION DESCRIBES THE JOB OUTPUT SPOOL PORTION.
*
*****
CPDURTN 347 4 U 4
CPLOGUN 351 3 U 0
CPPAGES 354 3 U 0
CPDEVNM 357 8 A
CPRROUTE 365 2 B MASK (HEX)
CPFORM 367 4 A
*****
*
* THIS SECTION DESCRIBES THE USER PORTION.
*
*****
USERAREA 371 254 A
*****
*
* THIS SECTION DESCRIBES THE SCRATCH PAD PORTION.
*
*****
*
CXPRLIN 646 4 U 2
CXPENCRD 650 4 U 2
CXURCST 654 5 U 2
*
CXCPUSRB 659 4 U 4
CXCPUCB 663 4 U 4
CXCPUCST 667 6 U 2
*
CXIOTAPE 673 4 U 2
CXIODSK1 677 4 U 2
CXIODSK2 681 4 U 2
CXIODSKX 685 4 U 2
CXI00TH 689 4 U 2
CXIODSKT 772 5 U 2
CXIOCST 773 5 U 2
*
CXCORUSE 703 2 B 0
CXCORALL 705 2 B 0
CXCORTOT 707 2 B 0
*
CXUNTAPE 709 3 U 2
CXUNDSK1 712 3 U 2
CXUNDSK2 715 3 U 2
CXUNDSKX 718 3 U 2
CXUN0TH 721 3 U 2
CXUNDISK 724 4 U 2
CXUNCST 728 4 U 2
*
CXWGTPTY 732 2 U 2
CXWGTCPU 734 2 U 2
*
CXPERTGET 736 3 U 2
CXPERTPUT 739 3 U 2
CXTPGCST 742 4 U 2
CXCONCHG 746 4 U 2
```

```

*
FCLASS      754      12      A      HEADING ('JOB' 'CLASS')
TITLE1      754      10      A      HEADING ('TIME' 'INTERVAL')
TITLE2      754      14      A      HEADING ('CPU TIME' 'INTERVAL')
SERVUNIT    754      20      A      HEADING ('SERVICE' 'UNITS')
DEPTKEY     37       4       A
DEPTRES     804     31      A
DEPTFLD     804     20      A
DERROR      804     16      A
ERRDEPT     820     4       A
CREDKEY     37       2       A
CREDRES     844     33      A
CREDFLD     844     9       N 2
BUDGFLD     854     9       N 2
MONTKEY     926     2       N 0
MONTFLD     884     10     A      HEADING ('MONTH')
DEBIT       940     5       U 2
OVRDATE     106     1       U
DATE1       930     5       N
DATE2       935     5       N
OUTPG       1004    4       N      HEADING ('PER.' 'GROUP')
JIF11SRT    1008    1       A
PRTZERO     1009    2       P      MASK ('ZZ9') +
      HEADING ('NUMBER' 'TAPES' 'ALLOCATED')
CXTOTIO     1011    4       U 2
*****
*
* THIS SECTION DESCRIBES THE WORKING STORAGE FIELDS *
* USED BY THE SAMPLE REPORTS SUPPLIED WITH THIS *
* FACILITY. *
*****
CNT01       W       5       N 0      HEADING ('NUM.' 'OF' 'JOBS')
CNT02       W       5       N 0      HEADING ('NUM.' 'OF' 'SES.')
JOBCNT01    S       5       P 0      HEADING ('NUM.' 'OF' 'JOBS')
JOBCNT02    S       5       P 0      HEADING ('NUM.' 'OF' 'SES.')
JOBCNT03    S       5       P 0
JOBCNT04    S       5       P 0
JOBCNT06    S       5       P 0
JOBCOUNT    S       5       P 0
TOTCPU05    S       7       P 4
TOTCPU      S       7       P 4
DSKTOT08    W       5       P 0      HEADING ('DISK I/O' 'COUNT')
WORKDATE    W       8       A
DISKTOT     W       5       P 0      HEADING ('TOTAL' 'I/O' 'COUNT')
IOTOT       W       5       P 0      HEADING ('TOTAL' 'I/O' 'COUNT')
SWPCNT      W       5       P 0      HEADING ('TOTAL' 'PAGEIN' 'PAGEOUT')
PAGES       W       5       P 0      HEADING ('TOTAL' 'PAGES')
JPERCENT    W       6       N 3      HEADING ('PERCENT' 'EXECS')
SORTFLD     W      10      A
STJOB       W       8       A
DEBTOT      S       5       U 2
TOTACT      S       8       P 4      HEADING ('GRAND')
AVGCPU      W       4       P 4      HEADING ('AVG' 'CPU' 'TIME')
AVGACT      W       5       P 4      HEADING ('AVG' 'ACTIVE' 'TIME')
AVGRES      W       5       U 4      HEADING ('AVG' 'RESIDENT' 'TIME')
AVGELAP     W       5       P 4      HEADING ('AVG' 'CONNECT' 'TIME')
PCTACT      W       3       P 3      HEADING ('PCT' 'OF' 'TOTAL')
JOBNO       W       2       U 0      HEADING ('TSU' 'NO.')
HOLDKEY     S       4       A
HOLDKEY2    HOLDKEY 2       A
HOLDDESC    S      33      A
HOLDCRED    HOLDDESC 9       N 2
HOLDBUDG    HOLDDESC +10 9       N 2
DBTCHG      W       6       P 2      MASK ('ZZZ,ZZZ,ZZZ.99-')
CRTAMT      W       6       P 2      MASK ('ZZZ,ZZZ,ZZZ.99-')
TOTCHG      W       6       P 2      MASK ('ZZZ,ZZZ,ZZZ.99-')

```

BGTAMT	W	6	P	2	MASK ('ZZZ,ZZZ,ZZZ.99-')
BGTDEV	W	6	P	2	MASK ('ZZZ,ZZZ,ZZZ.99-')
BGTPCT	W	6	P	2	MASK ('ZZZ,ZZZ,ZZZ.99-')
PRT_TALLY_	S	6	N	0	MASK ('ZZZZZ9-')
WORK_2N_	S	2	N	0	
B_FLAG_	S	1	A		VALUE ('B')
C_FLAG_	S	1	A		VALUE ('C')
T_FLAG_	S	1	A		VALUE ('T')
Y_FLAG_	S	1	A		VALUE ('Y')
ZERO_	S	4	P	0	VALUE (0)
ONE_	S	4	P	0	VALUE (1)
TWO_	S	4	P	0	VALUE (2)
REPORT_FLAG_	S	3	A		VALUE ('NO')
YES_	S	3	A		VALUE ('YES')
NO_	S	3	A		VALUE ('NO')
N100_	S	4	P	0	VALUE (100)
N1000_	S	4	P	0	VALUE (1000)
N1999_	S	4	P	0	VALUE (1999)
N2000_	S	4	P	0	VALUE (2000)
N2999_	S	4	P	0	VALUE (2999)
N3000_	S	4	P	0	VALUE (3000)
N3999_	S	4	P	0	VALUE (3999)
N4000_	S	4	P	0	VALUE (4000)
N4999_	S	4	P	0	VALUE (4999)
N5000_	S	4	P	0	VALUE (5000)
N9999_	S	4	P	0	VALUE (9999)
N10000_	S	4	P	0	VALUE (10000)
N19999_	S	4	P	0	VALUE (19999)
N20000_	S	4	P	0	VALUE (20000)
N29999_	S	4	P	0	VALUE (29999)
N30000_	S	4	P	0	VALUE (30000)
N39999_	S	4	P	0	VALUE (39999)
N40000_	S	4	P	0	VALUE (40000)
N49999_	S	4	P	0	VALUE (49999)
N50000_	S	4	P	0	VALUE (50000)
N74999_	S	4	P	0	VALUE (74999)
N75000_	S	4	P	0	VALUE (75000)
N99999_	S	4	P	0	VALUE (99999)
N100000_	S	4	P	0	VALUE (100000)

## Consolidated Record Field Descriptions

The following table describes each field name defined in the JIFREC macro:

Field Name	Description
XFLAG1	Indicates presence of main job section. "Y" = job section present.
XFLAG2	Indicates presence of step portion. "Y" = step section present.
XFLAG3	Indicates presence of spool section. "Y" = spool section present.
XFLAG4	Reserved
XFLAG5	Reserved
XFLAG6	Reserved
XFLAG7	Reserved
XFLAG8	Indicates presence of user section. "Y" = user section present.

Field Name	Description
COMSOURCE	Source of this job information. "S" = SMF.
CMFLAG1	<p>State of this record:</p> <p>C = This record is complete, such as all SMF records required to create this record have been processed.</p> <p>B = This record is complete, but was modified by the user. All SMF records required to create this record were processed, but the user in some way modified the record prior to its being written to the consolidated file.</p> <p><b>Note:</b> Modification does not include addition of the user portion to the record.</p> <p>• = This record is an orphan, or in other words, all SMF records required to create this record have not been processed.</p> <p>X = This record is an orphan and was modified by the user. All SMF records required to create this record have not been processed, and the user in some way modified the record before it was written to the consolidated file.</p> <p><b>Note:</b> Modification does not include addition of the user portion to the record.</p>
CMFLAG2	<p>Information source for this record:</p> <p>T = This record reports on TSO information.</p> <p>B = This record reports on batch job information.</p> <p>R = This record reports on batch information and is the product of a rerun situation.</p>
CMFLAG3	<p>User segment present indicator:</p> <p>Blank = The user segment is not present in this record.</p> <p>U = The user segment is present in this record.</p>
CMFLAG4	Reserved
CMFLAG5	Reserved
CMFLAG6	Reserved
CMFLAG7	Reserved
CMTOTSTP	Number of steps in this job
CMOFFSTP	Reserved
CMTOTSPL	Number of spool records associated with this job
CMOFFSPL	Reserved
CMCOUNT1	Reserved
CMCOUNT2	Job start date in packed, Julian format (YYDDD)
CMSYSID	Job identification
CMJOBNM	Job name
CMJOBNO	Job number

<b>Field Name</b>	<b>Description</b>
CMJOBPT	Job priority
CMJOBCL	Job class
CMPRKEY	Protect key of job
CMPRGNAM	Programmer's name field from job statement
CMUSRID	User identification
CMACCT	Accounting information from job statement
CMABEND	Job abend indicator "Y" = a job step abend
CMDTSTT	Job start date
CMTMSTT	Job start time
CMTMSTP	Job stop time
CMTMELAP	Job elapsed time, in minutes to four decimal places
CMTMCPU	Job CPU time, in minutes to four decimal places
CMTMTCB	Job CPU time under a Time Control Block (TCB), in minutes to four decimal places
CMTMSRB	Job CPU time under a Service Request Block (SRB), in minutes to four decimal places (OS/390 and above only)
CMTMACT	Active time, in minutes to four decimal places
CMTMRES	Resident time, in minutes to four decimal places
CMTMALD	Allocation delay times, in seconds to two decimal places
CMTMRDR	Time job was on input queue, in minutes to four decimal places
CMTMWRQ	Time job was on output queue, in minutes to four decimal places
CMTMWTR	Writer duration time, in minutes to four decimal places
CMTMTURN	Turnaround time, in minutes to four decimal places
CMCARDR	Number of cards read by job
CMCARDP	Number of cards punched by job
CMILNES	Number of lines printed by job
CMTAPE	Number of tapes used by job
CMDISK1 MDISK2 CMDIXKX	Number of disks used by job (in categories defined in JIFOPTS)
CMOTHER	Number of nondisk or nontape devices used by job

<b>Field Name</b>	<b>Description</b>
CMIOTP	EXCP count for tapes
CMIODK1 CMIODK2 CMIODKX	EXCP count disk devices (in categories defined in JIFOPTS)
CMIOOTH	EXCP count for other devices
CMVIOIN	I/O count for VIO-ins
CMVIOOT	I/O count for VIO-outs
CMSWAPIN	I/O count for swap-ins
CMSWAPOT	I/O count for swap-outs
CMPAGEIN	I/O count for page-ins
CMPAGEOT	I/O count for page-outs
CMADDSP	Number of address space swap sequences
CMPGSEC	Number of page seconds
CMSERV	Number of service units
CMPERFGP	Performance group number
CMCOREAL	Amount of core allocated for job, in 1 KB units
CMCOREUS	Amount of core used by job, in 1 KB units
CMTSOTG	The number of lines of terminal input (number of TGETS satisfied)
CMTSOTP	Number of lines of terminal output (number of TPUTS issued)
DATE	Overlay of start date field (CMDTSTT) to extract month
STRTTM	Overlay of start time field (CMTMSTT) to extract hour
CNTTAPE	Overlay of number of tape fields (CMTAPE) for a standard report
CSNAME	Step name
CSPROGN	Name of program executed in this (CSNAME) step
CSCOREAL	Amount of core allocated for step, in 1 KB units
CSCOREUS	Amount of core used in step, in 1 KB units
CSABND	Abend code if this step abended (Sxxx = system abend; otherwise, user code)
CSSTDT	Step start date
CSSTTM	Step start time
CSSPTM	Step stop time
CSELAPT	Step elapsed time, in minutes to four decimal places

<b>Field Name</b>	<b>Description</b>
CSCPU	CPU time for step, in minutes to four decimal places
CSTCB	CPU time for step under TCB, in minutes to four decimal places
CSSRB	CPU time for step under SRB, in minutes to four decimal places
CSACT	Active time for step, in minutes to four decimal places
CSRES	Resident time for step, in minutes to four decimal places
CSALDEL	Allocation delay time, in seconds to two decimal places
CSTAPNUM	Number of tapes used in step
CSDISK1 CSDISK2 CSDISKX	Number of disks used by step according to categories defined in the options macro JIFOPTS
CSOTHNM	Number of nondisk or tape devices used in step
CSIOTOP	EXCP count for tapes in CSOTHNM
CSIODK1 CSIODK2 CSIODKX	EXCP count for disk devices used in CSOTHNM, in categories defined in options macro JIFOPTS
CSIOOTH	EXCP count for nondisk or tape devices used in this step
CSVIOIN	I/O count for VIO-ins
CSVIOOT	I/O count for VIO-outs
CSSWAPIN	I/O count for swap-ins
CSSWAPOT	I/O count for swap-outs
CSPAGEIN	I/O count for page-ins
CSPAGEOT	I/O count for page-outs
CSADDSP	Number of address space swap sequences
CSPGSEC	Number of page seconds
CSSERV	Number of service units
CSTSOTG	Number of lines of terminal input (number of TSGETS satisfied)
CSTSOTP	Number of lines of terminal output (number of TPUTS issued)
CPDURTN	Writer duration time, in minutes to four decimal places
CPLOGUN	Number of logical records written
CPPAGES	Number of pages of output produced
CPDEVNM	Device name

<b>Field Name</b>	<b>Description</b>
CPROUTE	Route codes
CPFORM	Forms identification
USERAREA	Defines the user section as one field SCRATCH PAD AREA
CXURCST	Result field of unit record cost calculation
CXCPUSRB	Cost of SRB CPU time for this record
CXCPUTCB	Cost of TCB CPU time for this record
CXIOTAPE	Cost of tape I/O
CXIODSK1 CXIODSK2 CXIODSKX	Cost of disk I/O categorized according to definitions in options macro JIFOPTS
CXIOOTH	Cost of other devices I/O activity
CXIODSKT	Total cost of disk I/O
CXIOCST	Total cost of all I/O applicable to this record
CXCORUSE	Cost of core used
CXCORALL	Cost of core allocated
CXCORTOT	Total cost related to core usage
CXUNTAPE	Cost related to tape unit allocation
CXUNDSK1 CXUNDSK2 CXUNDSKX	Cost related to disk unit. Usages by categories defined in the options macro JIFOPTS
CXUNOTH	Cost related to unit usage not defined previously
CXUNDISK	Total cost related to disk unit usage
CXUNCST	Total cost of all unit usage
CXWGTPTY	Cost weighted because of certain job priority
CXWGTCPU FCLASS	Cost weighted because of certain CPT usage
TITLE1 TITLE2 SERVUNIT	Used in certain reports as CONTROL fields
DEPTKEY	Key field for department tape lookup
DEPTRES	Result field for department table lookup
DEPTFLD	Overlay of result field for report printing
CREDKEY	Key field for credit table lookup

Field Name	Description
CREDRES	Result field for department table lookup
CREFLD	Overlay of result field
BUDGFLD	Overlay of result field
MONTKEY	Key field for month table lookup
MONTFLD	Result field for month tape lookup
OVRDATE	Redefine date field to extract year
DEBIT	Work field in debit calculations
DATE1 DATE2	Work fields for picking up date parameters

### Audit File

One audit file record is created each time JIFSEL is executed. The record contains four segments:

- A static portion where various count information is stored
- A data set section
- One IPL section for each SMF type-00 record processed
- One lost data section for each SMF type-07 record processed

A report on the contents of the record is produced by JIFSEL after its processing is completed.

The layout of the audit file record is shown in Audit Record Fields.

### Audit Record Fields

The following lists the field definitions of the audit file. Descriptions of the field names are shown in the table that follows. An asterisk indicates breaks between the four record segments described on the previous page.

FILE AUDIT				
AUDDATE	1	3	U	MASK ('99/99/99')
AUDTIME	4	3	U	MASK ('99:99:99')
AUSMFDTF	7	3	U	MASK ('99/99/99')
AUSMFTMF	10	3	U	MASK ('99:99:99')
AUSMFDTL	13	3	U	MASK ('99/99/99')
AUSMFTML	16	3	U	MASK ('99:99:99')
AUCMDTF	19	3	U	MASK ('99/99/99')
AUCMTMF	22	3	U	MASK ('99:99:99')
AUCMDTL	25	3	U	MASK ('99/99/99')
AUCMTML	28	3	U	MASK ('99:99:99')
AUSMREAD	31	5	P	MASK ('ZZZZZZZ9')
AUSMRJCT	36	5	P	MASK ('ZZZZZZZ9')

AUSMFRCD	41	5	P	MASK ('ZZZZZZZ9')
AUCMCREA	46	4	P	MASK ('ZZZZZZ9')
AUCMDEL	50	4	P	MASK ('ZZZZZZ9')
AUCMMOD	54	4	P	MASK ('ZZZZZZ9')
AUCMORPH	58	4	P	MASK ('ZZZZZZ9')
AUCMRRUN	62	4	P	MASK ('ZZZZZZ9')
AUSMDUP	66	4	P	MASK ('ZZZZZZ9')
AUSDUMY	70	1	A	
AUOFFSM0	71	2	B	
AUOFFSM7	73	2	B	
AUDSTOT	75	2	B	
AUSM0TOT	77	2	B	MASK ('ZZZZ9')
AUSM7TOT	79	2	B	MASK ('ZZZZ9')
* VS_ OCCURS FOR THE LENGTH OF THE LONGEST SINGLE SEGMENT				
VS_	81	1	A	OCCURS 50
VS1_	VS_	50	A	INDEX SUB1
AUSDSNAM	VS1_	44	A	
AUDSVOL	VS1_ +44	6	A	
*				
VS2_	VS_	12	A	INDEX (SUB1, SUB2)
AUSM0SID	VS2_	4	A	
AUIPLDT	VS2_ +04	3	U	MASK ('99/99/99')
AUIPLTM	VS2_ +07	3	U	MASK ('99:99:99')
AUSM0PT	VS2_ +10	1	B	
AUSM0XX	VS2_ +11	1	A	
*				
VS3_	VS_	18	A	INDEX (SUB1, SUB2, SUB3)
AUSM7SID	VS3_	4	A	
AUTLOST	VS3_ +04	2	B	
AULSTDT	VS3_ +06	3	U	MASK ('99/99/99')
AULSTTM	VS3_ +09	3	U	MASK ('99:99:99')
AULSFDT	VS3_ +12	3	U	MASK ('99/99/99')
AULSFTM	VS3_ +15	3	U	MASK ('99:99:99')

## Audit Record Field Descriptions

The following table describes each field name in the audit record:

Field Name	Description
<b>Static Section</b>	
AUDATE	Date of run
AUTIME	Time of run
AUSMFDTF	Date of first SMF input record
AUSMFTMF	Time of first SMF input record
AUSMFDTL	Date of last SMF input record
AUSMFTML	Time of last SMF input record
AUCMDTF	Date of first consolidated record output
AUCMTFMF	Time of first consolidated record output
AUCMDTL	Date of last consolidated record output

<b>Field Name</b>	<b>Description</b>
AUCMTML	Time of last consolidated record output
AUSMREAD	Number of SMF records read
AUSMRJCT	Number of SMF records rejected
AUSMFRCD	Number of SMF records forced at EXIT1
AUCMCREA	Number of consolidated records created
AUCMDEL	Number of consolidated records deleted by user exit
AUCMMOD	Number of consolidated records modified by user exit
AUCMORPH	Number of consolidated records that are orphan records
AUCMRRUN	Number of rerun condition records
AUSMDUP	Number of duplicate data records
AUSDUMY	Reserved
AUOFFSMO	Offset of IPL section from start of record
AUOFFSM7	Offset of data lost section from start of record
AUDSTOT	Number of data set entries in record
AUSM0TOT	Number of IPL entries in record
AUSM7TOT	Number of data lost entries in record
<b>Data Set Selection</b>	
AUDSNAM	Data set name of input file to PANJOB
AUDSVOL	Volume serial number of AUDSNAM data set
<b>IPL Section</b>	
AUSMOSID	System identification of CPU experiencing IPL
AUIPLDT	Date of IPL
AUIPLTM	Time of IPL
AUSMOPT	SMF options in effect during execution of driver program
AUSM7SID	System identification of CPU from which data was lost
AUTLOST	Number of lost SMF records
AULSTDT	Starting date for lost records
AULSTTM	Starting time for lost records
AULSFDT	Finishing date for lost records
AULSFTM	Finishing time for lost records

## Audit Report

The audit report is based on the audit record and is produced for each execution of JIFSEL. The report contains record counts, first and last dates and times of the SMF records processed, IPL information, and lost data information.

The following shows an example of the audit report produced by JIFSEL:

```

03/22/84      COMPUTER ASSOCIATES - OS JOB INFORMATION FACILITY

PART ONE  ** INPUT  **

-A-  DATASET PROCESSING      DATASET NAME  MONDAY.SMFRECS.DATA
      VOLUME SERIAL  WORK01

-B-  DATE/TIME              FIRST SMF RECORD   03/19/84   08:08:23
      LAST SMF RECORD   03/19/84   22:13:28

-C-  RECORD TOTALS          SMF RECORDS READ      28226
      SMF RECORDS REJECTED  22211
      SMF RECORDS FORCED    0
      SMF RECORDS DUPLICATE  23

-D-  # I.P.L. RECORDS      1      DATE      TIME
      03/19/84   10:14:24

-E-  # DATA LOST RECORDS  0

PART TWO  ** OUTPUT **

-A-  RECORD TOTALS          CONSOLIDATED RECORDS  CREATED      881
      DELETED      0
      ORPHAN      174
      RERUN      0
      MODIFIED     0

-B-  DATE/TIME              FIRST CONSOLIDATED RECORD   03/16/84   20:23:41
      LAST CONSOLIDATED RECORD  03/20/84   07:34:21

```

## User Exit Facility

The user exit facility allows you to select and process all SMF records not automatically processed by JIFSEL. There are two entry points in the user exit facility. Each has a specific function:

**EXIT1**— Allows you to code your own routines to select any SMF records not processed automatically. The EXIT1 facility is an extension of JIFSEL's record selection process.

**EXIT2**— Is used to process the records selected by your EXIT1 routine. This consists of extracting data selected by the EXIT1 routine and inserting it into the User Area section provided in the consolidated record. The EXIT2 facility is an extension of JIFSEL's record consolidation function. This allows you to customize the record written to the consolidated file.

## EXIT1

The supplied EXIT1 default (JIFEXIT1) has no effect on the SMF record selection performed by the driver program. JIFEXIT1 is a one-instruction program and simply returns to JIFSEL each time it is called.

To process SMF record types not provided in JIFSEL or the JIF options table, you must write a program and substitute it for the default. See the [Installation Guide](#) for details about how to write your EXIT1 routine.

## EXIT2

The supplied EXIT2 default routine (JIFEXIT2) has no effect on the contents of the consolidated file or on any of the functions performed by JIFSEL. JIFEXIT2 is a one-instruction program and simply returns to JIFSEL each time it is called.

Four events cause EXIT2 to be invoked:

- Duplicate records.
- Rerun records.
- The presence of a record type unknown to JIFSEL (a record forced at EXIT1).
- The consolidated record is to be written.

Each class of event is discussed separately. Your routine is written to accommodate each situation.

See the [Installation Guide](#) for details about how to write your EXIT2 routine.

## Read Input Exit

JIF is a two-step process:

1. Produces the consolidated file
2. Toolkit routines are invoked to produce the reports (see [Facility Operation](#) earlier in this chapter)

JIFRDREX is the name of the read input exit that is used to read the consolidated file. JIFRDREX reads the consolidated file, then presents a fixed-length record to Toolkit for reporting.

## Using the Exit

JIFRDREX presents, to the routines, only the information you request. For example, job information, job and step information, job and user-appended information. Your request for information is based on parameters that are passed to JIFRDREX.

In the following table, parameters 1 through 8 indicate the information you want to process from the consolidated file:

Parameters	Options
1	Y = Return job information. N = Job information not required.
2	Y = Return step information. N = Step information not required.
3	Y = Return spool information. N = Spool information not required.
4-7	N = Required. These flags are reserved for future use.
8	Y = If present, return user-generated information. N = User information not required.

## Using the Availability Flags

JIFRDREX communicates with the routines through the first eight bytes of the record (the fields XFLAG1 through XFLAG8). These fields act as availability flags and indicate whether job, step, and spool detail are present on the record.

To indicate the level of detail, specify the appropriate parameter on the macro invocation statement of JIFREC. The numbers represent positional parameters and identify which information is requested.

JIFRDREX sets the availability flags to Y (yes) or N (no) to indicate the presence of the requested information.

## XFLAG Definitions

When JIFRDREX reads records from the consolidated file, it sets the XFLAG fields (bytes 1 through 8) to indicate the record contents. The fields are defined in the following table:

For XFLAG	The Meaning/ Content is:	JIFRDREX Exit Sets the Flags to:
1	New job information.	Y = When the job information fields in a record have changed to include new information. N = When the contents of the job information fields have not changed.
2	New step information.	Y = When step information was requested, and the content of the step information fields in a record changed. N = When step information was not requested, is not present, or has been exhausted for this job.
3	New spool information.	Y = When spool information was requested, and the contents of the spool information field in a record changed. N = When spool information was not requested, is not present, or has been exhausted for this job.
4-7	Reserved for future use.	N = These flags are not currently being used. They are always set to N.
8	New user.	Y = When user information was requested and was present in the record. N = When user information was not requested, is not present, or had been previously formatted for this job.

## Example

To demonstrate what is returned to the routines when processing the consolidated file, consider the following example.

Input

JIFOPTS specifies the following parameters:

```
STEPICD=YES; SPOLICD=YES;
EXIT2=USEREXIT.
```

USEREXIT is your routine that appends the 256-position user area to all nonorphan consolidated records.

The consolidated job record produced from the SMF data contains information for three steps and two spooled data sets, as well as the user-added section.

The following statement precedes the invocation of the JIF report:

```
%JIFREC YYNNNNY
```

The only job record in the consolidated record file is the one described previously.

#### Output

```
XFLAG      CONTENTS OF CONSOLIDATED RECORD FIELDS (Letters=prefixes)
FIELDS

          RECORD 1

1 = Y      CM - Accumulated job information
2 = Y      CS - Information on the first step executed in the job stream
3 = Y      CP - Information on the first spooled data set
8 = Y      User Portion - 256-byte user section

          RECORD 2

1 = N      CM - Same as in record 1
2 = Y      CS - Information on second execution in the job stream
3 = Y      CP - Information on the second spooled data set
8 = N      User Portion - Same as record 1

          RECORD 3

1 = N      CM - Same as record 1
2 = Y      CS - Information on the third step executed in the job stream
3 = N      CP - same as record 2
8 = N      User Portion - Same as record 1
```

## Statistical Reporting Routines

The OS Job Information Facility (JIF) provides reporting routines and an audit report routine that produce preformatted reports from the SMF data in the consolidated file.

The routines bypass all orphan and user-modified consolidated records. (Appending a section to an otherwise unaltered consolidated record is not considered modification in this context.)

## Syntax

To execute a JIF routine, you must invoke two macros. The following is the basic format:

```
%JIFREC      YYY  Y
              NNNNNNNN
%OSJIFxx     Startdate  Enddate
```

The JIFREC macro contains the file and field definitions for the contents of the consolidated file.

The Y/N (Yes/No) identifiers refer to the availability flags (XFLAGS) located at bytes 1 through 8 of the consolidated record. They indicate the type of information from the consolidated record that is to be made available to the JIF statistical routine. Where:

Y Indicates that a type of information is to be made available

N Indicates that it is not available.

See [Read Input Exit](#) earlier in this chapter.

The OSJIFxx macro identifies the reporting routines, where xx represents routines 01 through 19.

Startdate and Enddate are expressed in Julian format (YYDDD). They refer to the selected job information reporting period for each of the routines except where noted. All parameters for all JIF reports are required.

Unless units of time are explicitly identified in a column heading of a statistical report (for example, PAGE SECONDS in report OSJIF10), all units of time shown in the reports are in minutes to four decimal places.

In columns labeled AVE ALLOCATION TIME (reports OSJIF04, OSJIF09, OSJIF11, OSJIF13), units of time are reported in seconds, to two decimal places.

## Building Customized Routines

The supplied routines are general and may not always meet your needs. You can write your own routines either by modifying an existing routine or by designing new routines using Toolkit.

## JCL Example

The following JCL example shows an example which assumes that the consolidated file is created in a previous step. The DEPTTAB file is required only for the OSJIF14 routine.

```
//jobname      JOB      accounting.info
//STEPNAME     EXEC     PGM=EZTPA00
//STEPLIB      DD
//SYSPRINT     DD      SYSOUT=A
//SYSUDUMP     DD      SYSOUT=A
//SYSOUT       DD      SYSOUT=A
//PANDD        DD      DSN=PAPL.macro.library,DISP=SHR
//CONSOL       DD      DSN=OSJIF.consol.idated.data,DISP=SHR
//AUDIT        DD      DSN=OSJIF.audit.data.DISP=SHR
//EZTVFM       DD      UNIT=SYSDA,SPACE=(4096,(100,100))
//SORTWK01     DD      UNIT=SYSDA,SPACE=(CYL,5)
//SORTWK02     DD      UNIT=SYSDA,SPACE=(CYL,5)
//SORTWK03     DD      UNIT=SYSDA,SPACE=(CYL,5)
//DEPTTAB      DD      *
               -----
               DEPARTMENT TABLE
               -----
//SYSIN        DD      *
%JIFREC YNNNNNNN
%OSJIFxx yyddd yyddd
```

## Statistical Routines

This section describes the statistical routines.

### Service Unit Distribution - OSJIF01

The OSJIF01 routine generates a report summarizing the service units provided in general support of all processing within the range of dates you specify. The data is sequenced by OS/390 or z/OS service unit ranges in an OS/390 or z/OS performance group.

#### Syntax

```
%OSJIF01 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following is an example of the OSJIF01 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF01 89354 89354
```

Output

```

                                SERVICE UNIT DISTRIBUTION (ROUTINE OSJIF01)
                                DATA SELECTED BETWEEN DATES - 89354 AND 89354
                                PAGE 1

PER.  SERVICE          PERCENT  NUMBER  SERVICE  AVG  AVG  TOTAL
GROUP  UNITS            EXECS   OF JOBS  UNITS    SRB  TCB  I/O
                                I/O    TIME  TIME  COUNT  SECONDS  SWAP
                                COUNT

0001  100000 OR GREATER  100.000  6  1,050,969  .1134  .1871  15,142  3
OPER. GROUP SUBTOTAL  100.000  6  1,050,969  .1134  .1871  15,142  3

                                100.000  6  1,050,969  .1134  .1871  15,142  3

```

Report Contents

The following table describes each field name in the OSJIF01 report:

Field Name	Description
Performance Group	OS/390 or z/OS performance group
Service Unit Types	Range of OS/390 or z/OS service units
Percent Execs	Percentage of exec instructions performed that fall in the service unit range
Number of Jobs	Number of jobs selected from the input file that fall in the service unit range
Service Units	Number of service units recorded by OS/390 or z/OS for jobs that fall in the specified dates
Average SRB Time	Average time spent under control of a Service Request Block (SRB) for the service unit range
Average TCB Time	Average time spent under control of a Task Control Block (TCB) for the service unit range
Total I/O Count	Total input/output requests of all jobs in the service unit range

Field Name	Description
Page Seconds	Total number of page seconds for all jobs in the service unit range
Swap Count	Total numbers of address space swap sequences for jobs in the service unit range

## Performance Objective Summary - OSJIF02

The OSJIF02 routine generates a summary service distribution report on SMF data recorded between the specified dates. The data is sequenced by the performance objective of your choice in a performance group.

### Syntax

```
%OSJIF02 startdate enddate objective
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

objective

Any field described in the consolidated job record can be considered as a performance objective, with the exception of the Scratch pad and working storage field areas.

Although any field can be used, most cannot produce meaningful groupings. The following is a list of suggested performance objective fields:

Field Name	Description
CMCOUNT2	Group by julian date (YYYYDD)
CMSYSID	Group by system ID
CMJOBNM	Group by job name
CMJOBPT	Group by job priority
CMJOBCL	Group by job class
CMPRGNAM	Group by programmer name
CMUSRID	Group by user ID

### Example

The following sections show an example of the OSJIF02 report. The performance objective is Programmer Name. The report contents are also described.

#### Input

```
%JIFREC YNNNNNNN
%OSJIF02 89354 89354 CMPRGNAM
```

#### Output

PERFORMANCE OBJECTIVE SUMMARY (ROUTINE OSJIF02)										PAGE	1
DATA SELECTED BETWEEN DATES - 89354 AND 89354											
PER. GROUP	PER. OBJECTIVE	PERCENT	OF EXEC	NUM. OF JOBS	NUM. OF SERVICE SES.	RESIDENT UNITS	CPU TIME	TOTAL I/O TIME	PAGE COUNT	TOTAL PAGEIN SECONDS	PAGEOUT
0001	DATA.CENTER	BKUPS	100.000	6		1,050,969	67.2843	1.8038		15,142	6
SUBTOTAL			100.000	6		1,050,969	67.2843	1.8038		15,142	6
			100.000	6		1,050,969	67.2843	1.8038		15,142	6

#### Report Contents

The following table describes each field name in the OSJIF02 report:

Field Name	Description
Performance Group	OS/390 or z/OS performance group
Performance Objective	User-selected performance objective
Percent Execs	Percentage of jobs that fall in the specified group
Number of Jobs	Number of jobs that fall in the specified group
Number of Sessions	TSO sessions that fall in the specified group
Service Units	Number of service units recorded by OS/390 or z/OS for jobs that fall in the specified group
Resident Time	Total time the transactions within the specified group remained in real storage
CPU Time	Total combined time the transactions in the specified group spent under control of a Task Control Block and Service Request Block
Total I/O Count	Total input/output requests of all jobs in the specified group
Page Seconds	Total number of page seconds for all jobs in the specified group
Total Page-in Page-out	Total nonswap page-ins and page-outs for jobs in the specified group



Field Name	Description
CPU Time	Total combined time the transactions that occurred in the given month spent under control of a Task Control Block and Service Request Block
Service Units	Number of service units recorded by OS/390 or z/OS for jobs that fall in the given month
Total I/O Count	Total disk and tape input/output events that occurred for all jobs in the given month
Lines Printed	Total number of lines produced by all jobs in the given month
Percent Execs	Percentage of the total number of jobs that fall in the given month

## Performance Group Profile - OSJIF04

The OSJIF04 routine generates a summarized performance group profile report for a selected period of time and deals with average time profiles.

### Syntax

```
%OSJIF04 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following sections show an example of the OSJIF04 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF04 89354 89354
```

Output

```

                                PERFORMANCE GROUP PROFILE (ROUTINE OSJIF04)
                                DATA SELECTED BETWEEN DATES - 89354 AND 89354
                                PAGE 1

```

PER. OF GROUP	NUM. OF JOBS	NUM. OF SES.	AVG RDR TIME	AVG ALLOCATION TIME	AVG ACTIVE TIME	AVG RESIDENT TIME	AVG CPU TIME	AVG WRITER TIME	AVG ELAPSED TIME
0001	6		500.6669	2.00	11.4660	11.2140	.3006	.0000	11.8509
	6		500.6669	2.00	11.4660	11.2140	.3006	.0000	11.8509

## Report Contents

The following table describes each field name in the OSJIF04 report:

Field Name	Description
Performance Group	OS/390 or z/OS performance group
Number of Jobs	Number of jobs selected from the input file that fall in the service unit range
Number of Sessions	TSO sessions selected from the input file that fall in the performance group
Average Reader Time	Average time each job in the performance group spent on the reader queue
Average Allocation Time	Average allocation delay time for each job in the performance group
Average Active Time	Average time each job in the performance group is active
Average Resident Time	Average time each job in the performance group spent in real storage
Average CPU Time	Average combined time each job in the selected performance group spent under control of a Task Control Block and Service Request Block
Average Writer Time	Average time each job in the performance group spent in the output queue
Average Elapsed Time	Average elapsed time for each job in the performance group

## Performance Group Summary - OSJIF05

The OSJIF05 routine provides a performance group summary report for a selected period of time and deals with resource requirements in performance groups.

### Syntax

```
%OSJIF05 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following sections show an example of the OSJIF05 report. The report contents are also described.

#### Input

```
%JIFREC YNNNNNNN
%OSJIF05 89354 89354
```

#### Output

PERFORMANCE GROUP SUMMARY (ROUTINE OSJIF05)										PAGE	1
DATA SELECTED BETWEEN DATES - 89354 AND 89354											
PER. OF	NUM. OF	SERVICE	SWAP	TCB	SRB	CPU	ACTIVE	RESIDENT	PERCENT		
GROUP	JOB	SES. UNITS	COUNT	TIME	TIME	TIME	TIME	TIME	EXECS		
0001	6	1,050,969	3	1.1231	.6807	1.8038	68.7964	67.2843	100.000		
	6	1,050,969	3	1.1231	.6807	1.8038	68.7964	67.2843	100.000		

#### Report Contents

The following table describes each field name in the OSJIF05 report:

Field Name	Description
Performance Group	OS/390 or z/OS performance group
Number of Jobs	Number of jobs in the performance group
Number of Sessions	TSO sessions in the performance group
Service Units	Number of service units recorded by OS/390 or z/OS for jobs in the selected performance group
Swap Count	Total numbers of address space swap sequences for jobs in the performance group
TCB Time	Total time jobs in the performance group spent under control of a Task Control Block (TCB)
SRB Time	Total time jobs in the performance group spent under control of a Service Request Block (SRB)
CPU Time	Total amount of CPU time used by jobs in the performance group. CPU time = TCB time + SRB time.
Active Time	Total time jobs in the performance group are active
Resident Time	Total time jobs in the performance group spent in real storage
Percent CPU	Percentage of total CPU time used for jobs that fall in the selected performance group

## Performance Group/Priority/Class - OSJIF06

The OSJIF06 routine provides a report that calculates timing averages for jobs categorized by job class, job priority, and performance group.

### Syntax

```
%OSJIF06 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following sections show an example of the OSJIF06 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF06 89354 89354
```

Output

```

                                PERFORMANCE GROUP / PRIORITY / CLASS PROFILE (ROUTINE OSJIF06)                                PAGE      1
                                DATA SELECTED BETWEEN DATES - 89354 AND 89354

      P   NUM. NUM. AVG      AVG      AVG      AVG      AVG      AVG      AVG
      P   T C  OF  OF  RDR   WRITER  ELAPSED  TURNAROUND  SRB   TCB   CPU
      G   Y L  JOBS SES. TIME   TIME   TIME   TIME   TIME   TIME   TIME   TIME
0001   S   6           500.6669 .0000   11.8509   .0000   .1134   .1871   .3006
           6           500.6669 .0000   11.8509   .0000   .1134   .1871   .3006

```

Report Contents

The following table describes each field name in the OSJIF06 report:

Field Name	Description
Performance Group (PG)	OS/390 or z/OS performance group
Priority (PTY)	Priority of the job
Class (CL)	Job class
Number of Jobs	Number of jobs that fall in the performance group/job class group

Field Name	Description
Number of Sessions	TSO sessions that fall in the performance group/job class group
Average Reader Time	Average time each job in the performance group spent on the reader queue
Average Writer Time	Average time each job in the performance group spent in the output queue
Average Elapsed Time	Average elapsed time for each job in the performance group
Average Turnaround Time	Average turnaround time for each job in the performance group
Average SRB Time	Average time jobs in the performance group spent under control of a Service Request Block (SRB)
Average TCB Time	Average time jobs in the performance group spent under control of a Task Control Block (TCB)
Average CPU Time	Average combined time each job in the selected performance group spent under control of a Task Control Block (TCB) and Service Request Block (SRB)

## Hourly Throughput Analysis - OSJIF07

The OSJIF07 routine provides an hour-by-hour throughput analysis on a day-by-day basis.

### Syntax

```
%OSJIF07 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following sections show an example of the OSJIF07 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF07 89354 89354
```

Output

HOURLY THROUGHPUT ANALYSIS (ROUTINE OSJIF07)  
 DATA SELECTED BETWEEN DATES - 89354 AND 89354  
 FOR 12/20/89

PAGE 1

TIME INTERVAL	NUM. OF JOBS	NUM. OF SES.	AVG ACTIVE TIME	AVG RESIDENT TIME	AVG CPU TIME	SRB TIME	CPU TIME	SERVICE UNITS	PAGE SECONDS
00 - 01 AM	6		11.4660	11.2140	.3006	.6807	1.8038	1,050,969	15,142
	6		11.4660	11.2140	.3006	.6807	1.8038	1,050,969	15,142
	6		11.4660	11.2140	.3006	.6807	1.8038	1,050,969	15,142

Report Contents

The following table describes each field name in the OSJIF07 report:

Field Name	Description
Time Interval	Time frame during the day in which the jobs are started
Number of Jobs	Number of jobs that are started during the specified time interval
Number of Sessions	TSO sessions that are started during the specified time interval
Average Active Time	Average time each job in the time interval is active
Average Resident Time	Average time each job in the time interval remained in real storage
Average CPU Time	Average CPU time consumed by each job in the given time interval
SRB Time	Total time jobs initiated in the time interval spent under control of a Service Request Block (SRB)
CPU Time	Total CPU time consumed by each job in the given time interval
Service Units	Number of service units recorded by OS/390 or z/OS for jobs in the selected time interval
Page Seconds	Total number of page seconds for all jobs in the time interval

### Service Requirements - OSJIF08

The OSJIF08 routine provides a detailed service requirements report. Each job, for the selected period of time, produces a line entry on the report. Information is reported in job name sequence.

#### Syntax

```
%OSJIF08 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following sections show an example of the OSJIF08 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF08 89354 89354
```

Output

SERVICE REQUIREMENTS (ROUTINE OSJIF08)											PAGE	1
DATA SELECTED BETWEEN DATES - 89354 AND 89354												
											P	
JOB NAME	PER. GROUP	C L	T Y	SERVICE UNITS	TCB TIME	SRB TIME	DISK I/O COUNT	TAPE I/O COUNT	PAGE SECONDS	PAGEIN COUNT	PAGEOUT COUNT	
DBUSR053	1	S		174,180	.1783	.1078			2,523			
DBUSR054	1	S		221,818	.2255	.1535			3,079	5		
DBUSR055	1	S		114,786	.1441	.0738			2,015			
DBUSR056	1	S		220,431	.2118	.1633			2,707			
DBUSR057	1	S		108,196	.1393	.0620			1,621			
DBUSR058	1	S		211,558	.2241	.1203			3,197	1		
				1,050,969	1.1231	.6807			15,142	6		

Report Contents

The following table describes each field name in the OSJIF08 report:

Field Name	Description
Job Name	Taken from the job statement
Performance Group	OS/390 or z/OS performance group
Class (CL)	Job class
Priority (PTY)	Priority of the job
Service Units	Number of service units recorded by OS/390 or z/OS for this job
TCB Time	Time a job spent under control of a Task Control Block (TCB)
SRB Time	Time a job spent under control of a Service Request Block (SRB)
Disk I/O Count	Total number of disk input/output events that occurred during the execution of a job
Tape I/O Count	Total number of tape input/output events that occurred during the execution of a job

Field Name	Description
Page Seconds	Total number of page seconds for this job
Page-in Count	Number of input/output page-ins recorded for this job
Page-out Count	Number of input/output page-outs recorded for this job

## Hourly Turnaround - OSJIF09

The OSJIF09 routine generates an analysis based on the average amount of time required to process a job in a given one-hour time period. A one-page report is produced for each day that occurred in the selected time frame defined by startdate and enddate. Data is sequenced by hourly period.

### Syntax

```
%OSJIF09 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following sections show an example of the OSJIF09 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF09 89354 89354
```

Output

HOURLY TURNAROUND (ROUTINE OSJIF09)										PAGE	1
DATA SELECTED BETWEEN DATES - 89354 AND 89354											
FOR 12/20/89											
TIME	NUM.	NUM.	AVG	AVG	AVG	AVG	AVG	AVG	PERCENT		
INTERVAL	OF	OF	RDR	ALLOCATION	ACTIVE	ELAPSED	CPU	WRITER	EXECES		
	JOBS	SES.	TIME	TIME	TIME	TIME	TIME	TIME			
00 - 01 AM	6	500.	6669	2.00	11.4660	11.8509	.3006	.0000	100.000		
	6	500.	6669	2.00	11.4660	11.8509	.3006	.0000	100.000		
	6	500.	6669	2.00	11.4660	11.8509	.3006	.0000	100.000		

Report Contents

The following table describes each field name in the OSJIF09 report:

<b>Field Name</b>	<b>Description</b>
Time Interval	One-hour time frame in which the job is started on the date selected
Number of Jobs	Number of jobs started during the time interval
Number of Sessions	TSO sessions started during the time interval
Average Reader Time	Average time each job in the time frame spent on the reader queue
Average Allocation Time	Average allocation delay time for each job in the time frame
Average Active Time	Average time each job in the time frame is active
Average Elapsed Time	Average elapsed time for each job in the time frame
Average CPU Time	Average combined time each job in the selected time frame spent under control of a Task Control Block (TCB) and Service Request Block (SRB)
Average Writer Time	Average time each job in the time frame spent in the output queue
Percent Execs	Percentage of the total number of jobs included in the report that began execution in the given time frame

## Page Peaking Periods

The OSJIF10 routine produces a report in which each working day is divided into two-hour time frames, and the jobs that started during those time frames are reported on.

### Syntax

```
%OSJIF10 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following sections show an example of the OSJIF10 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF10 89354 89354
```

Output

PEEK PAGING PERIODS (ROUTINE OSJIF10)										PAGE	1
DATA SELECTED BETWEEN DATES - 89354 AND 89354											
FOR 12/20/89											
TIME INTERVAL	NUM. OF JOBS	NUM. OF SES.	TOTAL I/O COUNT	AVG		ACTIVE TIME	RESIDENT TIME	ACTIVE TIME	RESIDENT TIME	PAGE SECONDS	TOTAL PAGES
				ACTIVE TIME	RESIDENT TIME						
000:00 TO 01:59	6			11.4660	11.2140	68.7964	67.2843	15,142		6	
	6			11.4660	11.2140	68.7964	67.2843	15,142		6	
	6			11.4660	11.2140	68.7964	67.2843	15,142		6	

Report Contents

The following table describes each field name in the OSJIF10 report:

Field Name	Description
Time Interval	Two-hour time frame in which the job is started on the date selected
Number of Jobs	Number of jobs started during the time interval
Number of Sessions	TSO sessions started during the time interval
Total I/O Count	Total disk and tape input/output events that occurred for all jobs that started in the given time interval
Average Active Time	Average time each job in the time frame is active
Average Resident Time	Average time each job in the time interval remained in real storage
Active Time	Total time all jobs in the time frame are active
Resident Time	Total time all jobs in the time interval remained in real storage
Page Seconds	Total number of seconds the jobs in the time frame held a page
Total Pages	Total number of pages used by the jobs in the time interval

## Class Structure Analysis - OSJIF11

The OSJIF11 routine produces a report that categorizes the jobs by job class. The routine assumes that the only valid classes are A through Z and 0 through 9.

### Syntax

```
%OSJIF11 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following sections show an example of the OSJIF11 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF11 89354 89354
```

Output

```

                                CLASS STRUCTURE ANALYSIS (ROUTINE OSJIF11)
                                DATA SELECTED BETWEEN DATES - 89354 AND 89354
                                PAGE      1

```

JOB CLASS	NUMBER OF JOBS	AVG ALLOCATION TIME	AVG ACTIVE TIME	AVG CPU TIME	CPU TIME	AVG TURNAROUND TIME	TAPE I/O COUNT	LINES PRINTED	PERCENT EXECS
CLASS S JOBS	6	2.00	11.4660	.3006	1.8038	.0000			100.000
	6	2.00	11.4660	.3006	1.8038	.0000			100.000

Report Contents

The following table describes each field name in the OSJIF11 report:

Field Name	Description
Job Class	The class in which the jobs reported on are executed
Number of Jobs	Number of jobs selected from the input data that fall in the given job class
Average Allocation Time	Average allocation delay time for jobs in the given job class
Average Active Time	Average time each job in the job class is active

Field Name	Description
Average CPU Time	Average CPU time for each job in the given class
CPU Time	Total CPU time for all jobs in the given class
Average Turnaround Time	Average time between the time the reader recognized the job and the time the last spooled data set (list or punch) is dispatched. If a job had no spooled output, this value will be zero in that job record.
Tape I/O Count	Total tape input/output events that occurred for all jobs in the given job class
Lines Printed	Total number of lines produced by all jobs in the given class
Percent Execs	Percentage of the total number of jobs that are executed in the given class

## CPU Time Distribution - OSJIF12

The OSJIF12 routine categorizes the jobs that occurred within the startdate and enddate into the following CPU time usage ranges:

```

0 - 5 seconds
5 - 30 seconds
30 - 60 seconds
1 - 2 minutes
2 - 5 minutes
5 - 10 minutes
10 - 30 minutes
30 - 60 minutes
over 1 hour

```

### Syntax

```
%OSJIF12 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

The following is an example of the OSJIF12 report. The report contents are also described.

Input

```

%JIFREC YNNNNNNN
%OSJIF12 89354 89354

```

Output

CPU TIME DISTRIBUTION (ROUTINE OSJIF12)  
DATA SELECTED BETWEEN DATES - 89354 AND 89354

PAGE 1

CPU TIME INTERVAL	NUMBER OF JOBS AND SESSIONS	CPU TIME	AVG CPU TIME	AVG SRB TIME	AVG TCB TIME	ACTIVE TIME	AVG ACTIVE TIME	SERVICE UNITS	PERCENT CPU
05 TO 30 SECS	6	1.8038	.3006	.1134	.1871	68.7964	11.4660	1,050,969	100.000
	6	1.8038	.3006	.1134	.1871	68.7964	11.4660	1,050,969	100.000

Report Contents

The following table describes each field name in the OSJIF12 report:

Field Name	Description
CPU Time Interval	CPU time usage ranges into which the jobs fell, based on CPU time
Number of Jobs and TSO Sessions	Number of jobs and TSO sessions that fell in the given CPU time range
CPU Time	Total CPU time for each time interval
Average CPU Time	Average CPU time for each job in the given range
Average SRB Time	Average time jobs in the given range spent under control of a Service Request Block (SRB)
Average TCB Time	Average time jobs in the given range spent under control of a Task Control Block (TCB)
Active Time	Total time all jobs in the time frame are active
Average Active Time	Average time each job in the time range is active
Service Units	Number of service units recorded by OS/390 or z/OS for all jobs that occurred in the given range
Percent CPU	Percentage of all CPU time that was used by all the jobs reported on in each range

### Tape Allocation - OSJIF13

The OSJIF13 routine provides a summarized report by job name grouped by number of tapes allocated to that job.

#### Syntax

```
%OSJIF13 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

**Example**

The following is an example of the OSJIF13 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF13 89354 89354
```

Output

TAPE ALLOCATION BY JOBNAME (ROUTINE OSJIF13)								PAGE 1	
DATA SELECTED BETWEEN DATES - 89354 AND 89354									
NUMBER TAPES ALLOCATED	JOB NAME	NUM. OF JOBS	NUM. OF SES.	AVG ALLOCATION TIME	AVG ELAPSED TIME	AVG CPU TIME	DISK TAPE I/O COUNT		
1	DBUSR053	1		1.97	12.9265	.2861			
1	DBUSR054	1		2.05	13.6956	.3790			
1	DBUSR055	1		1.94	6.4291	.2179			
1	DBUSR056	1		2.04	16.8840	.3751			
1	DBUSR057	1		1.98	7.5276	.2013			
1	DBUSR058	1		2.03	13.6431	.3444			
1		6		2.00	11.8509	.3006			
		6		2.00	11.8509	.3006			

Report Contents

The following table describes each field name in the OSJIF13 report:

Field Name	Description
Number Tapes Allocated	Number of tapes that are allocated to the job
Job Name	Job name
Number of Jobs	Number of times the job name being reported on occurred in the record selection dates
Number of Sessions	Number of TSO sessions that occurred in the record selection dates
Average Allocation Time	Average amount of allocation delay time for each execution of the job being reported on

Field Name	Description
Average Elapsed Time	Average amount of time consumed for each execution of the job being reported, from the time it started until execution finished
Average CPU Time	Average amount of CPU time used in each execution of the job being reported
Tape I/O Count	Total tape input/output events that occurred during job execution
Disk I/O Count	Total disk input/output events that occurred during job execution

## Application Trend Analysis - OSJIF14

When you invoke the OSJIF14 routine, job information is selected from the consolidated file and combined by month within a department. In the routine, the department key is the first four positions of the job name field (CMJOBNM) in the consolidated record.

This routine uses table processing to translate this key into an expanded department name.

### Syntax

```
%OSJIF14 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Creating the Department Table

For OSJIF14 to translate the department keys specified in the job accounting information, a department table, called DEPTTAB, must be created as follows:

```
Positions
 1 - 4   Department key
 5 - 35  Department name
36 - 80  Unused

//DEPTTAB DD *

ACCOACCOUNTING
MAILMAIL ROOM
MARKMARKETING
SHIPSHIPPING & RECEIVING
ETC.
```

Example

The following is an example of the OSJIF14 report. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF14 89353 89354
```

Output

```
APPLICATION TREND ANALYSIS (ROUTINE OSJIF14) PAGE 1
DATA SELECTED BETWEEN DATES - 89353 AND 89354
FOR DEPARTMENT - ACCOUNTING DEPT
```

MONTH	NUM. NUM.		ELAPSED TIME	SRB TIME	TCB TIME	TAPE		LINES PRINTED	PERCENT EXECS
	OF JOBS	OF SES.				SERVICE UNITS	DISK I/O		
DECEMBER	1		184.0383	.0095	.0515	27,005			.208
	1		184.0383	.0095	.0515	27,005			.208

```
APPLICATION TREND ANALYSIS (ROUTINE OSJIF14) PAGE 2
DATA SELECTED BETWEEN DATES - 89353 AND 89354
FOR DEPARTMENT - ADMINISTRATION
```

MONTH	NUM. NUM.		ELAPSED TIME	SRB TIME	TCB TIME	TAPE		LINES PRINTED	PERCENT EXECS
	OF JOBS	OF SES.				SERVICE UNITS	DISK I/O		
DECEMBER	15		16.5209	.3350	.9178	590,262		132	3.125
	15		16.5209	.3350	.9178	590,262		132	3.125

```
APPLICATION TREND ANALYSIS (ROUTINE OSJIF14) PAGE 3
DATA SELECTED BETWEEN DATES - 89353 AND 89354
FOR DEPARTMENT - ACQUISITION DEPT
```

MONTH	NUM. NUM.		ELAPSED TIME	SRB TIME	TCB TIME	TAPE		LINES PRINTED	PERCENT EXECS
	OF JOBS	OF SES.				SERVICE UNITS	DISK I/O		
DECEMBER	1		5.9751	.1348	.2056	142,952	147	9,122	.208
	1		5.9751	.1348	.2056	142,952	147	9,122	.208
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

```
APPLICATION TREND ANALYSIS (ROUTINE OSJIF14) PAGE 49
DATA SELECTED BETWEEN DATES - 89353 AND 89354
FOR DEPARTMENT - UNKNOWN DEPT. = YODE
```

MONTH	NUM. NUM.		ELAPSED TIME	SRB TIME	TCB TIME	TAPE		LINES PRINTED	PERCENT EXECS
	OF JOBS	OF SES.				SERVICE UNITS	DISK I/O		
DECEMBER	2		6.2008	.0401	.8046	462,897			.416
	2		6.2008	.0401	.8046	462,897			.416
	443	37	7,658.4790	13.6387	50.2666	33,659,840	6,478	31,811	100.000

Report Contents

The following table describes each field name in the OSJIF14 report:

Field Name	Description
Department	In-house department title, taken from DEPTTAB
Month	Period being reported on, taken from MONTTAB
Number of Jobs	Number of jobs that fall in the department and date being reported on
Number of Sessions	Number of TSO sessions that fall in the department and date being reported on
Elapsed Time	Total elapsed time for all jobs in the given period. This is defined as the difference between job start and end times.
SRB Time	Time the jobs spent under control of a Service Request Block (SRB)
TCB Time	Time the jobs spent under control of a Task Control Block (TCB)
Service Units	Number of service units recorded by OS/390 or z/OS for all jobs in the group
Tape/Disk I/O Count	Total disk and tape input/output events that occurred for all jobs in the given department
Number of Lines	Total number of lines produced by all jobs in the given department
Percent Execs	Percentage of the total number of jobs in the given department that are represented by the selected data

## Abends by Abend Code - OSJIF15

The OSJIF15 routine produces a report grouped by abend code that lists the job and step names which abnormally terminated during the period being reported on.

### Syntax

```
%OSJIF15 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

Example

On the following pages is an example of the OSJIF15 report. The report contents are also described.

Input

```
%JIFREC YNNNNNN
%OSJIF15 89353 89354
```

Output

ABENDS BY ABEND CODE (ROUTINE OSJIF15)										PAGE	1
DATA SELECTED BETWEEN DATES - 89353 AND 89354											
ABEND	PROGRAM	JOB	STEP	STEP	STEP	STEP	USER	STEP	STEP		
CODE	NAME	NAME	NAME	RUN	START	START	INFORMATION	ELAPSED	CPU		
				DATE	TIME	TIME		TIME	TIME		
S0C4	TESTPROG	SCHMIDJ	TESTPROG	89/17/33	36:17:33	-		340.0000	34.0000		
	TESTPROG	SCHMIDJ	TESTPROG	89/17/35	53:17:35	-		231.0000	33.0000		
ABEND CODE								571.0000	67.0000		
S222	UTL120	JAMESM	LOG	89/17/38	35:17:38	-		230.0000	10.0000		
	IEBGENER	JAMESM	GENER1	89/17/38	37:17:38	-		493.0000	38.0000		
	IEBGENER	JAMESM	GENER2	89/17/38	40:17:38	-		713.0000	32.0000		
	IFOX00	JAMESM	ASM	89/17/38	44:17:39	-		2,603.0003	10.0002		
	IEBUPDTE	JAMESM	UPDATE	89/17/39	00:17:39	-		601.0000	32.0000		
	UTL120	JAMESM	LOG	89/17/39	04:17:39	-		261.0000	10.0000		
	IEBGENER	JAMESM	GENER1	89/17/39	06:17:39	-		368.0000	30.0000		
	IEBGENER	JAMESM	GENER2	89/17/39	08:17:39	-		633.0000	30.0000		
	IFOX00	JAMESM	ASM	89/17/39	12:17:39	-		2,785.0003	8.0002		
	IEBUPDTE	JAMESM	UPDATE	89/17/39	29:17:39	-		548.0000	28.0000		
	UTL120	JAMESM	LOG	89/17/51	19:17:51	-		246.0000	10.0000		
	IEBGENER	JAMESM	GENER1	89/17/51	21:17:51	-		233.0000	26.0000		
	IEBGENER	JAMESM	GENER2	89/17/51	22:17:51	-		511.0000	28.0000		
	IFOX00	JAMESM	ASM	89/17/51	25:17:51	-		5,013.0004	26.0003		
	IEBUPDTE	JAMESM	UPDATE	89/17/51	56:17:51	-		485.0000	26.0000		
	IKJEFT01	SCHEDUL	PANSOPHI	89/22/35	13:22:38	-		9,266.0006	16.0005		
	IEV90	SCHMIDJ	ASM	89/17/35	45:17:35	-		890.0000	39.0000		
	IEWL	SCHMIDJ	LKED	89/17/35	50:17:35	-		425.0000	24.0000		
	TMSAUDIT	DAYUCC1A	STEPA	89/07/49	51:07:50	-		9,600.0001	4.0000		
	TMSCOPY	DAYUCC1A	STEP1	89/07/50	48:07:52	-		2,685.0006	91.0004		
	TMSEXPD	DAYUCC1A	STEP1A	89/07/52	05:07:52	-		3,760.0002	59.0001		
	TMSCYCLE	DAYUCC1A	STEP2	89/07/52	27:07:52	-		3,040.0001	86.0001		
	TMSCCTLG	DAYUCC1A	STEP3	89/07/52	46:07:53	-		6,863.0004	62.0003		
	TMSCLEAN	DAYUCC1A	STEP4	89/07/53	27:07:54	-		5,650.0004	3.0002		
	TMSRPT2	DAYUCC1A	UCC1TMS	89/07/54	01:07:54	-		4,993.0004	96.0001		
	TMSRPT3	DAYUCC1A	UCC1TMS	89/07/54	31:07:54	-		3,790.0002	23.0001		
	TMSRPT4	DAYUCC1A	UCC1TMS	89/07/54	54:07:55	-		3,140.0001	88.0000		
	TMSRPT6	DAYUCC1A	UCC1TMS	89/07/55	13:07:55	-		3,378.0002	9.0001		
	TMSCLEAN	DAYUCC1A	STEP11	89/07/55	33:07:55	-		2,696.0001	76.0000		
	PAN#2	FLAHERTV	BACKUP	89/08/18	47:08:30	-		6,501.0014	92.0009		
	PAN#2	FLAHERTV	BACKUP	89/08/30	27:08:34	-		6,550.0008	78.0006		
	FDRDSF	DBUSR054	DBFDR	89/00/01	27:00:13	-		5,033.0040	1.0023		
	IKJEFT01	LUTE	PRODLMP	89/07/22	35:15:30	-		2,911.0032	89.0028		
	IKJEFT01	LUTE2	PRODLMP	89/07/22	35:13:27	-		7,798.0138	.0117		
	IKJEFT01	PEARSON	PEARSON1	89/07/31	08:16:33	-		6,750.0253	78.0226		
	IKJEFT01	COVAS	COVAS	89/07/40	03:08:00	-		7,013.0061	99.0051		
	JTVMMSG	PEARSONC	INITMSG	89/07/40	26:07:40	-		61.0000	6.0000		
	PANEXEC	PEARSONC	LCSCSKO	89/07/40	27:07:41	-		7,443.0006	88.0006		
	JTVMMSG	PEARSONC	SCCSMSG	89/07/41	11:07:41	-		23.0000	5.0000		
	JTVMMSG	PEARSONC	FAILMSG	89/07/41	12:07:41	-		5.0000	.0000		
	JTVMMSG	PEARSONC	ABNDMSG	89/07/41	12:07:41	-		5.0000	.0000		
	JTVMMSG	PEARSONC	INITMSG	89/07/41	12:07:41	-		30.0000	6.0000		

PANEXEC	PEARSONC	LCSCHKO	89/07/41	13:07:41	-	7,693.0006	91.0006
JTVMMESG	PEARSONC	SCCSMSG	89/07/41	59:07:41	-	46.0000	5.0000
JTVMMESG	PEARSONC	FAILMSG	89/07/41	59:07:41	-	5.0000	.0000
JTVMMESG	PEARSONC	ABNDMSG	89/07/41	59:07:41	-	8.0000	.0000
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
CMFBATCH	PEARSON	PVDEL1	89/09/08	23:09:08	-	1,730.0001	3.0000
CMFBATCH	PEARSON	PVADD2	89/09/08	34:09:08	-	1,566.0000	96.0000
CMFBATCH	PEARSON	UPDCF3	89/09/08	43:09:08	-	2,396.0002	28.0002
IKJEFT01	PEARSON	ENDMSG	89/09/08	58:09:08	-	6.0000	.0000
IKJEFT01	PEARSON	FAILMSG	89/09/08	58:09:09	-	2,383.0001	39.0001
						664,287.5991	675.4644

### Report Contents

The following table describes each field name in the OSJIF15 report:

Field Name	Description
Abend Code	System or user-issued abend code
Program Name	Name of the program that ended abnormally
Job Name	Job name
Step Name	Step name
Step Run Date	Date on which the abended step began execution
Step Start Time	Time of day that the abended step began execution
User Information	Blank if no user information present
Step Elapsed Time	Length of time between the time the step began execution and the time it abended
Step CPU Time	Amount of CPU time consumed by the step execution prior to abending

## Abend by Program - OSJIF16

The OSJIF16 routine provides the same information as OSJIF15, with the exception that in OSJIF16 the information is presented by program name.

The report content explanations for this routine are identical to those listed in the OSJIF15 routine.

### Syntax

```
%OSJIF16 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

**Example**

An example of the OSJIF16 report follows. The report contents are also described in the OSJIF15 output.

Input

```
%JIFREC YNNNNNN
%OSJIF16 89353 89354
```

Output

ABNORMAL TERMINATIONS BY PROGRAM (ROUTINE OSJIF16)										PAGE	1
DATA SELECTED BETWEEN DATES - 89353 AND 89354											
PROGRAM NAME	JOB NAME	ABEND CODE	STEP NAME	STEP RUN DATE	STEP START TIME	USER INFORMATION	STEP ELAPSED TIME	STEP CPU TIME			
PROGRAM	AMASPZAP	SCHMIDJZ	SA03	STEP1	89/13/32	46:13:32	-	775.0000	34.0000		
								775.0000	34.0000		
PROGRAM	AMDPRDMP	MCREYNO1	SA03	PRDUMP2	89/11/51	09:11:52	-	7,700.0016	91.0013		
		MCREYNO1	SA03	PRDUMP2	89/15/10	09:15:11	-	2,875.0019	29.0014		
								10,575.0035	120.0027		
PROGRAM	APCS5102	COVAS2	SA03	APCS5102	89/08/25	42:08:26	-	3,238.0002	37.0002		
		COVAS2	SA03	APCS5102	89/09/11	39:09:12	-	3,941.0002	42.0002		
								7,179.0004	79.0004		
PROGRAM	APCS5320	COVAS1	SA03	APCS5320	89/14/26	04:14:26	-	8,606.0004	71.0004		
		COVAS1	SA03	APCS5320	89/14/32	40:14:33	-	8,570.0004	55.0004		
		COVAS1	S222	APCS5320	89/14/36	06:14:39	-	4,201.0046	24.0037		
		COVAS1	S222	APCS5320	89/14/45	01:14:49	-	5,176.0036	46.0030		
		COVAS1	SA03	APCS5320	89/14/50	37:14:51	-	9,605.0005	8.0004		
		COVAS1	S013	APCS5320	89/15/37	51:15:38	-	6,741.0004	41.0004		
		COVAS1	SA03	APCS5320	89/15/40	32:15:41	-	4,961.0004	13.0003		
		COVAS1	S013	APCS5320	89/15/42	35:15:43	-	2,093.0004	24.0003		
		COVAS1	S222	APCS5320	89/15/47	12:15:50	-	3,060.0054	88.0043		
		COVAS1	S013	APCS5320	89/15/53	04:15:53	-	5,773.0004	26.0003		
PROGRAM		COVAS1	S013	APCS5320	89/15/54	26:15:55	-	7,630.0004	70.0004		
								66,416.0169	466.0139		
PROGRAM	APCS5391	COVAS1	SA03	APCS5391	89/15/55	12:15:55	-	6.0000	.0000		
		COVAS1	SA03	APCS5391	89/15/53	39:15:53	-	8.0000	.0000		
		COVAS1	SA03	APCS5391	89/15/43	48:15:43	-	5.0000	.0000		
		COVAS1	SA03	APCS5391	89/15/41	02:15:41	-	170.0000	.0000		
		COVAS1	SA03	APCS5391	89/15/38	32:15:38	-	6.0000	.0000		
		COVAS1	SA03	APCS5391	89/14/51	35:14:51	-	223.0000	.0000		
		COVAS1	SA03	APCS5391	89/14/33	32:14:33	-	285.0000	.0000		
		COVAS1	SA03	APCS5391	89/14/26	56:14:26	-	171.0000	.0000		
								874.0000	.0000		

```

PROGRAM

```

IFOX00	LUTE1	SA03	ASM	89/10/59	51:11:00	-	2,451.0001	44.0001
	JAMESM	SA03	ASM	89/16/29	03:16:29	-	4,530.0001	67.0001
	JAMESM	SA03	ASM	89/16/32	20:16:32	-	3,541.0003	21.0002
	JAMESM	SA03	ASM	89/16/34	14:16:34	-	6,526.0003	58.0003
	JAMESM	SA03	ASM	89/16/22	14:16:22	-	1,966.0000	61.0000
	JAMESM	SA03	ASM	89/16/23	36:16:23	-	3,841.0002	82.0002
	JAMESM	SA03	ASM	89/16/25	39:16:25	-	1,253.0000	62.0000
	JAMESM	SA03	ASM	89/16/26	09:16:26	-	2,261.0001	19.0001
	JAMESM	SA03	ASM	89/16/27	30:16:27	-	2,726.0002	76.0002
	JAMESM	SA03	ASM	89/16/28	18:16:28	-	4,526.0003	11.0002
	DIRADSNT	SA03	ASM	89/17/07	05:17:10	-	7,830.0061	21.0056
	JAMESM	SA03	ASM	89/17/00	11:17:00	-	2,735.0003	13.0002
							664,287.5991*	9675.4644*

## Abend by Programmer - OSJIF17

The OSJIF17 routine provides the same information as OSJIF15 and OSJIF16 with the exception that in OSJIF17 the information is presented in order by programmer name.

The report content explanations for this routine are identical to those listed in the OSJIF15 routine.

### Syntax

```
%OSJIF17 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

An example of the OSJIF17 report follows. The report contents are described in the OSJIF15 output.

Input

```
%JIFREC YNNNNNNN
%OSJIF17 89353 89353
```

Output

ABEND BY PROGRAMMER (ROUTINE OSJIF17)  
 DATA SELECTED BETWEEN DATES - 89353 AND 89353

PAGE 1

PROGRAMMER NAME	JOB NAME	PROGRAM NAME	ABEND CODE	STEP NAME	STEP RUN DATE	STEP START TIME	STEP ELAPSED TIME	STEP CPU TIME
ADMIN	PRJPANBK	PAN#2	SA03	STEP1	89/17/59	04:18:00	8,588.0008	84.0006
	PRJPANBK	PAN#2	SA03	STEP2	89/18/00	55:18:01	8,853.0003	19.0002
	PRJPANBK	PAN#2	SA03	STEP3	89/18/01	48:18:01	93.0000	.0000
PROGRAMMER							17,534.0011	103.0008
AL TREVINO	TREVINO1	JTVMMESG	SA03	INITMSG	89/09/24	35:09:24	255.0000	6.0000
	TREVINO1	PANEXEC	SA03	LCSASMA	89/09/24	37:09:28	9,730.0023	58.0021
	TREVINO1	JTVMMESG	SA03	SCCSMSG	89/09/28	35:09:28	31.0000	5.0000
	TREVINO1	JTVMMESG	SA03	FAILMSG	89/09/28	35:09:28	5.0000	.0000
	TREVINO1	JTVMMESG	SA03	ABNDMSG	89/09/28	36:09:28	3.0000	.0000
PROGRAMMER							10,024.0023	69.0021
ARRAYCOMP PROGRAMMER	KADLCOMP	PAN#1	SA03	PANSTEP	89/10/26	41:10:26	636.0000	40.0000
							636.0000	40.0000
ARR007 PROGRAMMER	KINSR007	PAN#1	SA03	PANSTEP	89/11/22	22:11:22	598.0000	31.0000
							598.0000	31.0000
ARR008 PROGRAMMER	KINSR008	PAN#1	SA03	PANSTEP	89/13/52	20:13:52	2,170.0000	39.0000
							2,170.0000	39.0000
A2CICS17 PROGRAMMER	DIRADSNT	PAN#1	SA03	PAN	89/17/06	31:17:07	5,553.0001	52.0001
	DIRADSNT	IFOX00	SA03	ASM	89/17/07	05:17:10	7,830.0061	21.0056
	DIRADSNT	IEBUPDTE	SA03	BLDMBR	89/17/10	52:17:10	1,166.0000	61.0000
	DIRADSNT	IEWL	SA03	LNKEDT	89/17/10	59:17:11	1,796.0000	50.0000
PROGRAMMER							16,345.0062	184.0057
CHILDS PROGRAMMER	CHILDSMS	PAN#8	SA03	STEP1	89/11/58	12:11:59	8,181.0010	83.0009
							8,181.0010	83.0009
CLEANUP PROGRAMMER	GRYZIK	PAN#2	SA03	STEP1	89/10/22	31:10:22	2,876.0001	44.0001
							2,876.0001	44.0001
COPY EXCHANGE	SYSEXCH	IEBGENER	SA03	STEP1	89/10/17	22:10:20	7,075.0000	29.0000
	SYSEXCH	IEBGENER	SA03	STEP2	89/10/20	04:10:20	5,078.0000	26.0000
							12,153.0000	55.0000
. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .
. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .
. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .	. . . . .
012AD2TRAN PROGRAMMER	KINSTRAN	PAN#1	SA03	PANSTEP	89/10/15	41:10:15	828.0000	47.0000
							828.0000	47.0000
021AXXXX PROGRAMMER	HARTXXXX	PAN#1	SA03	PANSTEP	89/10/35	01:10:35	1,071.0000	99.0000
							1,071.0000	99.0000
065DDOSA PROGRAMMER	IWANDOSA	PAN#1	SA03	PANSTEP	89/11/06	29:11:06	1,101.0001	27.0001
	IWANDOSA	PAN#1	SA03	PANSTEP	89/11/02	27:11:02	820.0000	93.0000
PROGRAMMER							1,921.0001	120.0001
065DVICKLS PROGRAMMER	IWANCKLS	PAN#1	SA03	PANSTEP	89/15/39	09:15:39	845.0000	48.0000
							845.0000	48.0000
							150,163.3958*	2369.2953*

## Audit File Statistical Report - OSJIF18

The OSJIF18 routine provides the same accounting report as JIFSEL does, with the exception that OSJIF18 reports on all records in the audit file.

### Syntax

```
%OSJIF18
```

### Example

The following is an example of the OSJIF18 report. For an explanation of the report contents, see [JIFSEL Data Sets](#) and [Audit Report](#) earlier in this chapter.

#### Input

```
%OSJIF18
```

#### Output

```

                                COMPUTER ASSOCIATES - OS JOB INFORMATION FACILITY                                8.58.03
PART ONE ** INPUT **

-A-  DATASET PROCESSING          DATASET NAME          SUGEL.TEST.SMFDATA
                                VOLUME SERIAL        USR011

-B-  DATE/TIME                   FIRST SMF RECORD      12/20/89 00:51:04
                                LAST SMF RECORD       12/20/89 00:55:03

-C-  RECORD TOTALS              SMF RECORDS READ      22762
                                SMF RECORDS REJECTED  19365
                                SMF RECORDS FORCED    0
                                SMF RECORDS DUPLICATE 0

-D-  # I.P.L. RECORDS           0

-E-  # DATA LOST RECORDS        0

PART TWO ** OUTPUT **

-A-  RECORD TOTALS              CONSOLIDATED RECORDS  CREATED      854
                                DELETED      0
                                ORPHAN       373
                                RERUN        1
                                MODIFIED     0

-B-  DATE/TIME                   FIRST CONSOLIDATED RECORD 12/18/89 23:55:50
                                LAST CONSOLIDATED RECORD 12/19/89 22:35:13
    
```

## TSO Session Analysis - OSJIF19

The OSJIF19 routine does an analysis of TSO sessions by date in the range selected. This routine reports on TSO sessions only.

### Syntax

```
%OSJIF19 startdate enddate
```

startdate

The date (YYYYDD) that the selection of job information is to begin.

enddate

The date (YYYYDD) that the selection of job information is to end.

### Example

An example of the OSJIF19 report follows. The report contents are also described.

Input

```
%JIFREC YNNNNNNN
%OSJIF19 89353 89354
```

Output

TSO SESSION ANALYSIS (ROUTINE OSJIF19)												PAGE	1
DATA SELECTED BETWEEN DATES - 89353 AND 89354													
LOGON DATE: 12/19/89													
JOB	START	STOP	TSU	CONNECT	AVG	ACTIVE	PCT	AVG	CPU	AVG	TSO		
NAME	TIME	TIME	NO.	TIME	CONNECT	TIME	OF	ACTIVE	TIME	CPU	PUTS	GETS	
ADAMS	10:34:46	12:59:30	316	144.7213	144.7213	.9697	.411	.9697	.0341	.0341	160,000		
	16:34:55	17:05:29	1,017	30.5555	30.5555	.5652	.240	.5652	.0288	.0288	120,000		
JOBNAME	TOTALS		2	175.2768	87.6384	1.5349	.651	.7674	.0629	.0314	280,000		
BANYARD	8:52:05	11:56:08	95	184.0383	184.0383	1.3000	.552	1.3000	.0610	.0610	340,000		
JOBNAME	TOTALS		1	184.0383	184.0383	1.3000	.552	1.3000	.0610	.0610	340,000		
BAKER	10:10:34	10:15:59	232	5.4281	5.4281	.0963	.040	.0963	.0074	.0074	10,000		
JOBNAME	TOTALS		1	5.4281	5.4281	.0963	.040	.0963	.0074	.0074	10,000		
CHATMAN	11:02:51	13:28:40	379	145.8135	145.8135	4.6227	1.963	4.6227	.4094	.4094	790,000		
	13:28:54	16:43:58	558	195.0801	195.0801	10.6600	4.527	10.6600	1.2063	1.2063	840,000		
JOBNAME	TOTALS		2	340.8936	170.4468	15.2827	6.491	7.6413	1.6157	.8078	1,630,000		
DRAKE	7:40:03	8:00:45	19	20.7028	20.7028	7.8050	3.315	7.8050	.6199	.6199	770,000		
	8:00:54	12:16:46	34	255.8643	255.8643	20.6529	8.772	20.6529	1.9987	1.9987	2,520,000		
	13:28:43	16:03:28	557	154.7503	154.7503	24.4115	10.369	24.4115	1.4736	1.4736	3,690,000		
JOBNAME	TOTALS		3	431.3174	143.7724	52.8694	22.456	17.6231	4.0922	1.3640	6,980,000		
EDISON	13:50:09	13:52:44	582	2.5681	2.5681	1.5468	.657	1.5468	.0376	.0376	80,000		
JOBNAME	TOTALS		1	2.5681	2.5681	1.5468	.657	1.5468	.0376	.0376	80,000		

FISH	9:20:55	9:59:13	135	38.3033	38.3033	1.0702	.454	1.0702	.0456	.0456	30,000
JOBNAME TOTALS			1	38.3033	38.3033	1.0702	.454	1.0702	.0456	.0456	30,000
GRAVES	10:00:35	12:27:38	207	147.0578	147.0578	2.1470	.911	2.1470	.1613	.1613	300,000
	12:58:09	16:33:24	519	215.2548	215.2548	7.3301	3.113	7.3301	.4261	.4261	1,170,000
JOBNAME TOTALS			2	362.3126	181.1563	9.4771	4.025	4.7385	.5874	.2937	1,470,000
JONES	9:37:14	17:20:49	163	463.5750	463.5750	19.9872	8.489	19.9872	2.0402	2.0402	31,220,000
JOBNAME TOTALS			1	463.5750	463.5750	19.9872	8.489	19.9872	2.0402	2.0402	31,220,000
LARKIN	15:08:13	17:34:18	698	146.0848	146.0848	1.5589	.662	1.5589	.0911	.0911	310,000
JOBNAME TOTALS			1	146.0848	146.0848	1.5589	.662	1.5589	.0911	.0911	310,000
LARKIN1	12:16:07	15:01:44	463	165.6090	165.6090	1.4115	.599	1.4115	.1089	.1089	110,000
JOBNAME TOTALS			1	165.6090	165.6090	1.4115	.599	1.4115	.1089	.1089	110,000
MADISON	13:53:01	14:22:36	586	29.5795	29.5795	.8550	.363	.8550	.0586	.0586	280,000
JOBNAME TOTALS			1	29.5795	29.5795	.8550	.363	.8550	.0586	.0586	280,000
MERRILL	7:22:35	15:30:52	6	488.2928	488.2928	6.9268	2.942	6.9268	.3289	.3289	1,750,000
JOBNAME TOTALS			1	488.2928	488.2928	6.9268	2.942	6.9268	.3289	.3289	1,750,000
NORTON	7:22:35	13:27:22	12	364.7820	364.7820	17.4158	7.397	17.4158	1.3800	1.3800	7,280,000
	13:27:36	15:23:41	551	116.0895	116.0895	8.9512	3.802	8.9512	.4555	.4555	1,410,000
JOBNAME TOTALS			2	480.8715	240.4357	26.3670	11.199	13.1835	1.8355	.9177	8,690,000
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.
WALACE	8:05:47	10:43:46	44	157.9838	157.9838	.7664	.325	.7664	.0539	.0539	50,000
JOBNAME TOTALS			1	157.9838	157.9838	.7664	.325	.7664	.0539	.0539	50,000
ZEPHR	8:21:02	16:41:23	57	500.3430	500.3430	12.5884	5.347	12.5884	.6356	.6356	2,540,000
JOBNAME TOTALS			1	500.3430	500.3430	12.5884	5.347	12.5884	.6356	.6356	2,540,000
DATE TOTALS			37	6,260.5120	169.2030	235.4270	.000*	6.3628	17.1476	.4634	67,580,000
			37	6,260.5120	169.2030	235.4270	.000*	6.3628	17.1476	.4634	67,580,000

Report Contents

The following table describes each field name in the OSJIF19 report:

Field Name	Description
Job Name	The TSO session name
Start Time	Time the session started (on the 24-hour clock)
Stop Time	Time the session ended (on the 24-hour clock)
TSO No.	The TSO session number assigned at logon time
Connect Time	Length of time the user is connected
Average Connect Time	Average length of time the user is connected
Active Time	Total length of time the TSO user is active
PCT. of Total	Percentage of total time this TSO session is active

Field Name	Description
Average Active Time	Average length of time the TSO user is active
CPU Time	Number of CPU minutes used during the TSO session
Average CPU Time	Average number of CPU minutes used during the TSO session
TSO TPUTS	Number of TPUTS completed during the TSO session
TSO TGETS	Number of TGETS completed during the TSO session

## Weighting and Costing Examples

Weighting and costing examples contain sample Toolkit routines that demonstrate the use of the CA-Easytrieve Plus macro facility to generate weighting, costing, and billing information from the SMF data in the consolidated file.

Seven job accounting examples are shown on the following pages:

- Two for weighting:
  - Job Priority Weighting - EXWGTPTY
  - CPU Weighting - EXWGTCPU
- Five for costing:
  - CPU Charging - EXCHGCPU
  - I/O Charging - EXCHGIO
  - TPUT and TGET Charging - EXCHGTPG
  - Unit Record Device Charging - EXCHGUR
  - Combined Costs - EXCOSTS

These routines are supplied with the JIF and stored in your Toolkit macro library. They can be used without change or used as models for creating your own algorithms and routines.

Example Toolkit routines are shown later in this chapter to give you a feel for how billing routines can be created from the SMF data. The routines are:

- Invoice Ledger - JIFBEX01
- Detail Charge Audit - JIFBEX02
- Job Charge Summary - JIFBEX03
- Data Center Cost Recovery - JIFBEX04
- Monthly Utilization by Cost Center - JIFBEX05

- Data Processing Invoice - JIFBEX06
- TSO User Charging Report - JIFBEX07

**Note:** These examples are for demonstration only.

## Building Customized Routines

Any billing algorithm must be unique to the installation and company. This allows the user to alter the parameters of costs to suit changing overheads, hardware configurations, and other factors. For example, the actual charges vary, including the costs of tape versus disk I/O, core costs, unit amounts, CPU utilization, or execution class.

In addition, an installation may run a priority scheme. For example, a priority seven or higher job costs twice the basic amount, and priority nine and above cost three times the basic amount.

Also, in a multi-CPU environment, weighted charges may be required to control the cost of one second of CPU time on machines with differing speeds.

Considering these factors, there can be no universal definitive costing algorithm.

## Job Priority Weighting - EXWGTPTY

The EXWGTPTY routine compares the execution priority field from the job portion of the consolidated record (CMJOBPT) against a priority, then fills a work field (CXWGTPTY) with the weighting factor. CXWGTPTY is used in a later calculation to develop a cost.

### Syntax

```
%EXWGTPTY comparison priority weight
```

comparison

Indicates any valid Toolkit relational operator:

EQ	=		EQUAL TO
NE	≠	NQ	NOT EQUAL TO
LT	<	LS	LESS THAN
LE	<=	LQ	LESS THAN OR EQUAL TO
GT	>	GR	GREATER THAN
GE	>=	GQ	GREATER THAN OR EQUAL TO

priority

Indicates the priority that is to be tested.

weight

Indicates the weighting factor to be used if the condition tested for is true.

## Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
IF CMJOBPT &OPERATOR &PRIORITY
  CXWGTPY = &WEIGHT
END-IF
```

This routine must follow record qualification logic and precede cost calculation.

## Example

```
%EXWGTPY LE 7 1
%EXWGTPY GQ 8 2
%EXWGTPY GQ 10 2.5
```

In this example, the contents of CXWFGTPTY are 1.00 for jobs with a priority of 7 or less, 2.00 for jobs with a priority of 8 or 9, and 2.50 for jobs with a priority of 10 or higher.

## CPU Weighting - EXWGTCPU

The EXWGTCPU routine compares the system ID field (CMSYSID) from the consolidated record with the user-supplied SYSID parameter for an equal condition. If the comparison is true, then the weight parameter is moved to a work field (CXWGTCPU). CXWGTCPU is used in a later calculation to develop cost.

## Syntax

```
%EXWGTCPU sysid weight
```

sysid

A four-position alphanumeric constant that is to be compared to the system ID field (CMSYSID) in the consolidated record.

weight

The weighting to be used if the condition tested for is true.

## Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
IF CMSYSID = '&SYSID'  
  CXWGTCPU = &WEIGHT  
END-IF
```

This routine must follow record qualification logic and precede cost calculation.

## Example

```
%EXWGTCPU H158 1.0  
%EXWGTCPU 4341 1.5
```

In this example, jobs that executed on the 370/158 are weighted with a factor of 1.0 which represents basic cost. Jobs executing on the 4341 are weighted with a factor of 1.5 or one and one-half times the basic cost. This weighting factor is stored in CXWGTCPU.

## CPU Charging - EXCHGCPU

The EXCHGCPU routine performs two types of CPU utilization calculations:

- A calculation to generate a cost for each minute of Service Request Block (SRB) CPU utilization
- A calculation to generate a cost for each minute of Task Control Block (TCB) CPU utilization.

Both these charges are weighted by the CPU and priority weight factor fields. This routine yields three cost fields:

```
CXCPUSRB = SRB CPU Utilization  
CXCPUTCB = TCB CPU Utilization  
CXCPUCST = CXCPUSRB + CXCPUTCB
```

## Syntax

```
%EXCHGCPU    srbcost    tcbcost
```

srbcost

The charge unit for each minute of CPU utilization under a Service Request Block (SRB).

tcbcost

The charge unit for each minute of CPU utilization under a Task Control Block (TCB).

## Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
* CPU CHARGING ALGORITHM
* ARE WE PROCESSING STEPS
IF XFLAG2 = 'X'
  CXCPUSRB = CSSRB * &SRBCOST * CXWGTCPU * CXWGTPTY
  CXCPUTCB = CSTCB * &TCBCOST * CXWGTCPU * CXWGTPTY
  CXCPUCST = CXCPUSRB + CXCPUTCB
ELSE
  CXCPUSRB = CMTMSRB * &SRBCOST * CXWGTCPU * CXWGTPTY
  CXCPUTCB = CMTMTCB * &TCBCOST * CXWGTCPU * CXWGTPTY
  CXCPUCST = CXCPUSRB + CXCPUTCB
END-IF
```

**Note:** It is your responsibility to move an X to the field XFLAG2 before including the example CPU charging routine, if the CPU costs are being developed from steps.

## Example

```
%EXCHGCPU 7.45 9.9
```

In this example, SRB costs are developed at a rate of 7.45 charge units per minute of SRB time, and TCB costs are developed at a rate of 9.90 charge units per minute of TCB time.

## I/O Charging - EXCHGIO

The EXCHGIO routine develops I/O usage charges. This routine yields seven fields:

- CXIOTAPE=The number of tape I/O EXCPS multiplied by the tape charge unit
- CXIODSK1=The number of disk type 1 I/O EXCPS multiplied by the Disk1 charge unit
- CXIODSK2=The number of disk type 2 I/O EXCPS multiplied by the Disk1 charge unit
- CXIODSKX=The number of all other disk type I/O EXCPS multiplied by the DISKX charge unit

- CXIODSKT=CXIODSK1 + CXIODSK2 + CXIODSKX (total disk I/O charge)
- CXIOOTH=All non tape or disk EXCPS multiplied by the 'other' charge unit
- CXIOCST=CXIODSKT + CXIOTAPE + CXIOOTH multiplied by the CPU weighting factor relative to this record

## Syntax

```
%EXCHGIO    tape  disk1  disk2  diskx  other
```

tape

The tape EXCP I/O charge unit.

disk1

Disk type 1 (as defined by the DSKTYPE1 parameter of the JIFOPTS macro) EXCP I/O charge unit.

disk2

Disk type 2 (as defined by the DSKTYP2 parameter of the JIFOPTS macro) EXCP I/O charge unit.

diskx

The EXCP I/O charge unit for all other disk types.

other

The EXCP I/O charge unit for devices other than tape or disk drives.

## Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
* I/O CHARGING ALGORITHM
*   ARE WE PROCESSING STEP RECORDS
IF XFLAG2 = X
  CXIOTAPE = CSIOTP * &TAPE
  CXIODSK1 = CSIODK1 * &DISK1
  CXIODSK2 = CSIODK2 * &DISK2
  CXIODSKX = CSIODKX * &DISKX
  CXIODSKT = CXIODSK1 + CXIODSK2 + CXIODSKX
  CXIOOTH  = CSIOOTH * &OTHER
  CXIOCST  = (CXIODSKT + CXIOTAPE +CXIOOTH) * CXWGTCPU
ELSE
  CXIOTAPE = CMIOTP * &TAPE
  CXIODSK1 = CMIODK1 * &DISK1
  CXIODSK2 = CMIODK2 * &DISK2
```

```

CXIODSKX = CMIODKX * &DISKX
CXIODSKT = CXIODSK1 + CXIODSK2 +CXIDSKX
CXIOOTH = CMI00TH * &OTHER
CXIOCSST = (CXIODSKT + CXIOTAPE + CXIOOTH) * CXWGTCPU
END-IF

```

**Note:** It is your responsibility to move an X to the field XFLAG2 prior to this routine if I/O costs are being developed from step processing.

### Example

```
%EXCHGIO .14 .19 .19 .19 .10
```

In this example, tape EXCPS costs .14 charge units, all disk EXCPS costs .19 charge units, and all other EXCPS costs .10 charge units.

## TPUT and TGET Charging - EXCHGTPG

The EXCHGTPG routine calculates the cost of TGETS and TPUTS per TSO session and the total cost of TGETS and TPUTS.

### Syntax

```
%EXCHGTPG      tgetcost  tputcost
```

tgetcost

The charge unit for each TGET completed during the length of the session.

tputcost

The charge unit for each TPUT completed during the length of the session.

### Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```

F XFLAG1 EQ Y_FLAG_
  CXPERTGET = CMTSOTG * &TGETCOST
  CXPERTPUT = CMTSOTP * &TPUTCOST
  CXTPTGCST = CXPERTGET + CSPERTPUT
END-IF

```

## Example

```
%EXCHGTPG .20 .31
```

## Unit Record Device Charging - EXCHGUR

The EXCHGUR routine develops unit record usage cost, either the per line printed cost or per card punched cost. This routine yields three fields:

- CXPERLIN=The number of lines printed multiplied by the line charge unit
- CXPERCRD=The number of cards punched multiplied by the punched charge unit
- CXURCST=CXPERLIN + CXPERCRD

## Syntax

```
%EXCHGUR    lines    punched
```

lines

Indicates the lines per line printed charge unit.

punched

Indicates the per card punched charge unit.

## Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
IF XFLAG1 EQ Y_FLAG_
  CXPERLIN = CMLINES * &LINECOST
  CXPERCRD = CMCARDP * &CARD COST
  CXURCST = CXPERLIN + CXPERCRD
END-IF
```

## Example

```
%EXCHGUR .01 .05
```

In this example, each line printed costs .01 charge units, and each card punched costs .05 charge units.

## Combined Costs - EXCOSTS

The EXCOSTS routine demonstrates the ability to build a single macro to call the two weighting and three charging routines in order to build a billing algorithm.

### Syntax

```
%EXCOSTS      (none)
```

### Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
*
* SETUP WEIGHTINGS FOR PRIORITY
*
%EXWGTPY LE 7 1
%EXWGTPY GQ 8 2
%EXWGTPY GQ 9 3
*
* SETPU WEIGHTINGS FOR CPU
*
%EXWGTCPU H158 1.0
%EXWGTCPU 4341 1.5
*
* CALCULATE COSTS FOR I/O
*
%EXCHGIO .14 .19 .19 .19 .10
*
* NOW ADD TOTAL
*
%EXCHGUR .01 .05
DEBIT = CXCPUCST + CXIOCST
IF XFLAG1 EQ Y FLAG
    DEBIT = DEBIT + CXURCST
END-IF
*
* END OF EXCOSTS
```

**Note:** These routines are copies of the examples previously described.

One additional field is developed in this routine. DEBIT represents the total charge for the entire cost of the job.

## Billing Routines Examples

Seven example Toolkit routines (JIFBEX01 through 07) are shown on the following pages to demonstrate how billing routines can be created from the SMF data. They use the consolidated file and a billing algorithm to translate various statistical information into charges for use in billing or cost accounting.

**Note:** These examples are for demonstration only and are not a supported portion of the JIF utility.

### Syntax

```
%JIFREC      YNNNNNNN  
%JIFBEXnn   startdate  enddate
```

Where nn = the number 01 through 07.

startdate

The date (YYDDD) that selection of job information is to begin.

enddate

The date (YYDDD) that selection of job information is to end.

### Table Lookup

Use is made of two tables:

- DEPTTAB–Department table
- CREDITAB–Credit/Budget table

DEPTTAB

For several of the JIF billing examples to translate the department keys specified in the job accounting information, a department table, called DEPTTAB, must be created as follows:

```
Positions  
1 - 4      Department key  
5 - 35     Department name  
36 - 80    Unused
```

For example:

```
//DEPTTAB DD *  
ACCOACCOUNTING  
MAILMAIL ROOM  
SHIPSHIPPING & RECEIVING  
...  
/*
```

## CREDTAB

Several of the JIF billing examples allow crediting and budgeting of cost areas by using the first 2 characters of the department field as a key into a credit/budget table. In order for these examples to utilize this option, the credit/budget table, called CREDTAB, must be created as follows:

```
Positions
 1 - 2   Cost area key (First 2 characters of DEPTTAB)
 3 - 11  Credit amount for Cost area
12 - 12  Blank
13 - 21  Budget amount for Cost area
22 - 80  Unused
```

For example:

```
//CREDTAB DD *
AC001000000 001000000
MA000500000 000500000
SH001800000 001800000
...
/*
```

## Invoice Ledger - JIFBEX01

The JIFBEX01 routine demonstrates the ability to create an invoice ledger report.

### Syntax

```
%JIFREC      YNNNNNNN
%JIFBEX01    startdate  enddate
```

startdate

The date (YYDDD) that selection of job information is to begin.

enddate

The date (YYDDD) that selection of job information is to end.

### Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
IF CMFLAG1 EQ C_FLAG_
  IF CMCOUNT2 = &STARTDATE THRU &ENDDATE
    PERFORM COLLECT_DATA
    PRINT JIFBEX01
    REPORT_FLAG_ = YES_
  END-IF
END-IF
```

```
*
%JIFPROCS JIF BEX01
*
COLLECT_DATA. PROC
SEARCH DEPTTAB WITH DEPTKEY GIVING DEPTRES
IF NOT DEPTTAB
  DERROR = 'UNKNOWN DEPT. = '
  ERRDEPT = DEPTKEY
END-IF
*
%EXCOSTS
*
END-PROC
*

REPORT JIFBEX01 SUMMARY SUMCTL (HIAR) SPACE 1 SUMSPACE 2
SEQUENCE DEPTFLD
CONTROL FINAL DEPTFLD
TITLE 01 'INVOICE LEDGER (ROUTINE JIFBEX01)'
TITLE 02 'DATA SELECTED BETWEEN DATES - &STARTDATE AND &ENDDATE'
HEADING DEPTFLD ('COST CENTER')
HEADING DBTCHG ('DEBIT')
HEADING CRTAMT ('CREDIT')
HEADING TOTCHG ('TOTAL' 'CHARGE')
HEADING BGTAMT ('BUDGET')
HEADING BGTDEV ('OVER-' 'UNDER+' 'BUDGET')
HEADING BGTDPCT ('PERCENTAGE' 'OF' 'BUDGET')
LINE 01 DEPTFLD DBTCHG CRTAMT TOTCHG BGTAMT BGTDEV BGTDPCT
*
REPORT-INPUT. PROC.
IF DEPTKEY NE HOLDKEY
  HOLDKEY = DEPTKEY
  SEARCH CREDTAB WITH HOLDKEY2 GIVING HOLDDESC
  IF NOT CREDTAB
    HOLDCRED = 0
    HOLDBUDG = 0
  END-IF
ELSE
  HOLDCRED = 0
  HOLDBUDG = 0
END-IF
DBTCHG = DEBIT
CRTAMT = HOLDCRED
TOTCHG = DBTCHG - CRTAMT
BGTAMT = HOLDBUDG
BGTDEV = BGTAMT - TOTCHG
SELECT
END- PROC
*
BEFORE-BREAK. PROC
IF BGTAMT EQ 0
  BGTDPCT = BGTDEV * N100_
ELSE
  BGTDPCT = BGTDEV * N100_/ BGTAMT
END-IF
END-PROC
```

## Detail Charge Audit - JIFBEX02

The JIFBEX02 routine demonstrates the ability to create a detail charge audit report.

## Syntax

```
%JIFREC      YNNNNNNN
%JIFBEX02    startdate  enddate
```

startdate

The date (YYDDD) that selection of job information is to begin.

enddate

The date (YYDDD) that selection of job information is to end.

## Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing code.

```
IF CMFLAG1 EQ C_FLAG_
  IF CMCOUNT2 = &STARTDATE THRU &ENDDATE
    IF XFLAG2 EQ Y_FLAG_
      PERFORM COLLECT_DATA
      PRINT JIFBEX02
      REPORT_FLAG_ = YES_
    END-IF
  END-IF
END-IF
*
COLLECT_DATA. PROC
*
%JIFPROCS JIF BEX01
*
%EXCOSTS
*
DISKTOT = CSDISK1 + CSDISK2 + CSDISKX
IOTOT   = CSIODK1 + CSIODK2 + CSIODKX
END-PROC
*
REPORT JIFBEX02 SPACE 1 SUMSPACE 1
SEQUENCE CMJOBNM
CONTROL FINAL
TITLE 01 'DETAIL CHARGE AUDIT (ROUTINE JIFBEX02)'
TITLE 02 'DATA SELECTED BETWEEN DATES - &STARTDATE AND &ENDDATE'
HEADING CMJOBNM ('JOB' 'NAME')
HEADING CSPROGN ('PROGRAM' 'NAME')
HEADING CMJOBCL ('C' 'L' 'A')
HEADING CMJOBPT ('P' 'R' 'I')
HEADING CSCPU ('STEP' 'CPU' 'TIME')
HEADING CSIOTP ('TAPE' 'I/O')
HEADING IOTOT ('DISK' 'I/O')
HEADING CSTAPNUM ('#' 'TP')
HEADING DISKTOT ('#' 'DK')
HEADING CMLINES ('LINES' 'PRINTED')
HEADING MCARDR ('CARDS' 'READ')
HEADING MCARDP ('CARDS' 'PUNCHED')
HEADING DEBIT ('TOTAL' 'CHARGE')
LINE 01 CMJOBNM CSPROGN CMJOBCL CMJOBPT CSCPU CSIOTP IOTOT CSTAPNUM -
DISKTOT CMLINES MCARDR MCARDP DEBIT
```

## Job Charge Summary - JIFBEX03

The JIFBEX03 routine demonstrates the ability to create a job charge summary report.

### Syntax

```
%JIFREC      YNNNNNNN
%JIFBEX03    startdate enddate
```

startdate

The date (YYDDD) that selection of job information is to begin.

enddate

The date (YYDDD) that selection of job information is to end.

### Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
IF CMFLAG1 EQ C_FLAG
  IF CMCOUNT2 = &STARTDATE THRU &ENDDATE
    PERFORM COLLECT_DATA
    PRINT JIFBEX03
    REPORT_FLAG_ = YES_
  END-IF
END-IF
*
%JIFPROCS JIF BEX03
*
COLLECT_DATA. PROC
SEARCH DEPTTAB WITH DEPTKEY GIVING DEPTRES
IF NOT DEPTTAB
  DERROR = 'UNKNOWN DEPT. = '
  ERRDEPT = DEPTKEY
END-IF
*
%EXCOSTSIF CMFLAG1 EQ C_FLAG_
  IF CMCOUNT2 = &STARTDATE THRU &ENDDATE
    IF XFLAG2 EQ Y_FLAG_
      PERFORM COLLECT_DATA
      PRINT JIFBEX02
      REPORT_FLAG_ = YES_
    END-IF
  END-IF
END-IF
*
COLLECT_DATA. PROC
*
%JIFPROCS JIF BEX01
*
%EXCOSTS
```

```

*
DISKTOT = CSDISK1 + CSDISK2 + CSDISKX
IOTOT   = CSIODK1 + CSIODK2 + CSIODKX
END-PROC
*
REPORT JIFBEX02 SPACE 1 SUMSPACE 1
SEQUENCE CMJOBNM
CONTROL FINAL
TITLE 01 'DETAIL CHARGE AUDIT (ROUTINE JIFBEX02)'
TITLE 02 'DATA SELECTED BETWEEN DATES - &STARTDATE AND &ENDDATE'
HEADING CMJOBNM ('JOB' 'NAME')
HEADING CSPROGN ('PROGRAM' 'NAME')
HEADING CMJOBCL ('C' 'L' 'A')
HEADING CMJOBPT ('P' 'R' 'I')
HEADING CSCPU ('STEP' 'CPU' 'TIME')
HEADING CSIOTP ('TAPE' 'I/O')
HEADING IOTOT ('DISK' 'I/O')
HEADING CSTAPNUM ('#' 'TP')
HEADING DISKTOT ('#' 'DK')
HEADING CMLINES ('LINES' 'PRINTED')
HEADING CMCARDR ('CARDS' 'READ')
HEADING CMCARDP ('CARDS' 'PUNCHED')
HEADING DEBIT ('TOTAL' 'CHARGE')
LINE 01 CMJOBNM CSPROGN CMJOBCL CMJOBPT CSCPU CSIOTP IOTOT CSTAPNUM -
        DISKTOT CMLINES CMCARDR CMCARDP DEBIT

*
END-PROC
*
REPORT JIFBEX03 SPACE 1 SUMSPACE 2 DTLCTL EVERY SUMCTL NONE
SEQUENCE DEPTFLD
CONTROL FINAL DEPTFLD
TITLE 01 'JOB CHARGE SUMMARY (ROUTINE JIFBEX03)'
TITLE 02 'DATA SELECTED BETWEEN DATES - &STARTDATE AND ENDDATE'
HEADING DEPTFLD ('COST CENTER')
HEADING CMJOBNM ('JOBNAME')
HEADING TALLY ('NUMBER' 'OF' 'JOBS')
HEADING CXCPUCST ('PROCESS' 'CHARGES')
HEADING CXIOCST ('I/O' 'CHARGES')
HEADING CXURCST ('U/R' 'CHARGES')
HEADING DEBIT ('TOTAL' 'CHARGES')
LINE 01 DEPTFLD CMJOBNM TALLY CXCPUCST CXIOCST CXURCST DEBIT

```

## Data Center Cost Recovery - JIFBEX04

The JIFBEX04 routine demonstrates the ability to create a data center cost recovery report.

### Syntax

```

%JIFREC      YNNNNNNN
%JIFBEX04   startdate  enddate

```

startdate

The date (YYDDD) that selection of job information is to begin.

enddate

The date (YYDDD) that selection of job information is to end.

## Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
IF CMFLAG1 EQ C_FLAG_
  IF CMCOUNT2 = &STARTDATE THRU &ENDDATE
    PERFORM COLLECT_DATA
    PRINT JIFBEX04
    REPORT_FLAG_ = YES_
  END-IF
END-IF
*
%JIFPROCS JIF BEX04
*
COLLECT_DATA. PROC
SEARCH DEPTTAB WITH DEPTKEY GIVING DEPTRES
IF NOT DETPTAB
  DERROR = 'UNKNOWN DEPT. = '
  ERRDEPT = DEPTKEY
END-IF
*
%EXCOSTS
*
END-PROC
*
REPORT JIFBEX04 SPACE 1 SUMSPACE 2 DTLCTL EVERY SUMCTL NONE
SEQUENCE DEPTFLD
CONTROL FINAL DEPTFLD
TITLE 01 'DATA CENTER COST RECOVERY (ROUTINE JIFBEX04)'
TITLE 02 'DATA SELECTED BETWEEN DATES - &STARTDATE AND &ENDDATE'
HEADING DEPTFLD ('COST CENTER')
HEADING TALLY ('NUMBER' 'OF' 'JOBS')
HEADING CXPCUCST ('PROCESS' 'CHARGES')
HEADING CXIOCST ('I/O' 'CHARGES')
HEADING CXURCST ('U/R' 'CHARGES')
HEADING DEBIT ('TOTAL' 'CHARGES')
LINE 01 DEPTFLD TALLY CXPCUCST CXIOCST CXURCST DEBIT
```

## Monthly Utilization by Cost Center - JIFBEX05

The JIFBEX05 routine demonstrates the ability to create a monthly utilization by cost center report.

### Syntax

```
%JIFREC      YNNNNNNN
%JIFBEX05    startdate enddate
```

startdate

The date (YYDDD) that selection of job information is to begin.

enddate

The date (YYDDD) that selection of job information is to end.

### Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
IF CMFLAG1 EQ C_FLAG_
  IF CMCOUNT2 = &STARTDATE THRU &ENDDATE
    PERFORM COLLECT_DATA
    PRINT JIFBEX05
    REPORT_FLAG_ = YES_
  END-IF
END-IF
*
%JIFPROCS JIF BEX05
*
COLLECT_DATA. PROC
SEARCH DEPTTAB WITH DEPTKEY GIVING DEPTRES
IF NOT DEPTTAB
  DERROR = 'UNKNOWN DEPT. = '
  ERRDEPT = DEPTKEY
END-IF
*
%EXCOSTS
*
IOTOT = CMIOTP + CMIODK1 + CMIODK2 + CMIODKX + CMIOOTH
MONTKEY = DATE
SEARCH MONTHS WITH MONTKEY GIVING MONTFLD
END-PROC
*
REPORT JIFBEX05 SPACE 1 SUMSPACE 2 SUMMARY
SEQUENCE DEPTFLD CMCOUNT2
CONTROL FINAL DEPTFLD MONTFLD
TITLE 01 'MONTHLY UTILIZATION SUMMARY BY COST CENTER (ROUTINE
JIFBEX05)'
TITLE 02 'DATA SELECTED BETWEEN DATES - &STARTDATE AND &ENDDATE'
TITLE 03 'YEAR = 19' OVRDATE
HEADING DEPTFLD ('COST CENTER')
```

```
HEADING MONTFLD ('MONTH')
HEADING TALLY ('NUMBER' 'OF' 'JOBS')
HEADING CMTMELAP ('ELAPSED' 'TIME')
HEADING CMTMCPU ('CPU' 'TIME')
HEADING IOTOT ('TOTAL' 'I/O' 'COUNT')
HEADING DEBIT ('TOTAL' 'CHARGES')
LINE 01 DEPTFLD MONTFLD TALLY CMTMELAP CMTMCPU IOTOT DEBIT
```

## Data Processing Invoice - JIFBEX06

The JIFBEX06 routine demonstrates the ability to create a data processing invoice report.

### Syntax

```
%JIFREC      YNNNNNNN
%JIFBEX06    startdate enddate
```

startdate

The date (YYDDD) that selection of job information is to begin.

enddate

The date (YYDDD) that selection of job information is to end.

### Routine Code

The following Toolkit routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```
IF CMFLAG1 EQ C_FLAG_
  IF CMCOUNT2 = &STARTDATE THRU &ENDDATE
    PERFORM COLLECT_DATA
    PRINT JIFBEX06
    REPORT_FLAG_ = YES_
  END-IF
END-IF
*
%JIFPROCS JIF BEX06
*
COLLECT_DATA. PROC
SEARCH DEPTTAB WITH DEPTKEY GIVING DEPTRES
IF NOT DEPTTAB
  DERROR = 'UNKNOWN DEPT. = '
  ERRDEPT = DEPTKEY
END-IF
*
%EXCOSTS
*
END-PROC
*
REPORT JIFBEX06 SPACE 1 SUMSPACE 2 SUMMARY
SEQUENCE DEPTFLD
```

```

CONTROL FINAL DEPTFLD
TITLE 01 'DATA PROCESSING INVOICE (ROUTINE JIFBEX06)'
TITLE 02 'DATA SELECTED BETWEEN DATES - &STARTDATE AND &ENDDATE'
HEADING DEPTFLD ('COST CENTER')
HEADING DBTCHG ('DEBIT')
HEADING CRTAMT ('CREDIT')
HEADING TOTCHG ('TOTAL' 'CHARGE')
LINE 01 DEPTFLD DBTCHG CRTAMT TOTCHG
*
REPORT-INPUT. PROC
IF DEPTKEY NE HOLDKEY
  HOLDKEY = DEPTKEY
  SEARCH CREDTAB WITH HOLDKEY2 GIVING HOLDDESC
  IF NOT CREDTAB
    HOLDCRED = 0
  END-IF
ELSE
  HOLDCRED = 0
END-IF
DBTCHG = DEBIT
CRTAMT = HOLDCRED
SELECT
END-PROC
*
BEFORE-BREAK. PROC
TOTCHG + DBTCHG- CRTAMT
END-PROC
*

```

## TSO User Charging Report - JIFBEX07

The JIFBEX07 routine demonstrates the ability to create a TSO user charging report.

### Syntax

```

%JIFREC      YNNNNNNN
%JIFBEX07    startdate  enddate

```

startdate

The date (YYDDD) that selection of job information is to begin.

enddate

The date (YYDDD) that selection of job information is to end.

### Routine Code

The following CA-Easytrieve Plus routine is intended as an example to demonstrate how JIF creates a job accounting/billing system using Toolkit. As such, it is not part of the supported product, but serves as a model to assist you in developing job accounting/billing coding.

```

IF CMFLAG1 EQ C_FLAG_
  IF CMFLAG2 EQ T_FLAG_

```

```
        IF CMCOUNT2 = &STARTDATE THRU &ENDDATE
          PERFORM COLLECT_DATA
          PRINT JIFBEX07
          REPORT_FLAG_ = YES_
        END-IF
      END-IF
    END-IF
  *
  %JIFPROCS JIF BEX07
  *
  COLLECT_DATA. PROC
  *
  * SETUP WEIGHTINGS FOR PRIORITY
  *
  %EXWGTPTY LE 7 1
  %EXWGTPTY GQ 8 2
  %EXWGTPTY GQ 9 3
  *
  * SETUP WEIGHTINGS FOR CPU
  *
  %EXWGTCPU H158 1.0
  %EXWGTCPU 4341 1.5
  *
  * CALCULATE COSTS FOR CPU UTILIZATION
  *
  %EXCHGCPU 7.45 9.90
  *
  * CALCULATE COSTS FOR I/O
  %EXCHGIO .14 .19 .19 .19 .10
  *
  * CALCULATE COSTS FOR UNIT RECORD DEVICES
  *
  %EXCHGUR .01 .05
  *
  * CALCULATE COSTS FOR TGETS AND TPUTS
  *
  %EXCHGTPG .05 .01
  *
  * CALCULATE TOTALS AND REPORT SPECIFIC CHARGES
  *
  CXCONCHG = CXWGTCPU * CMTMELAP * 1.00
  DEBIT = CXCONCHG
  DEBIT = DEBIT + CXCPUCST
  DEBIT = DEBIT + CXIOCST
  DEBIT = DEBIT + CXURCST
  DEBIT = DEBIT + CXTPTGCST
  CXTOTIO = CXIOCST + CXURCST
  *
  END-PROC
  *
  REPORT JIFBEX07 SPACE 1 SUMSPACE 2 DTLCTL EVERY SUMCTL NONE
  SEQUENCE CMJOBNM CMCOUNT2 CMJOBNO
  CONTROL FINAL CMJOBNM
  TITLE 01 'USER CHARGING REPORT (ROUTINE JIFBEX07)'
  TITLE 02 'DATA SELECTED BETWEEN DATES - &STARTDATE AND &ENDDATE'
  HEADING CMJOBNM ('USER' 'ID')
  HEADING CMDTSTT ('LOGON' 'DATE')
  HEADING CMJOBNO ('TSU' 'NUMBER')
  HEADING CXCONCHG ('CONNECT' 'CHARGE')
  HEADING CXCPUCST ('PROCESS' 'CHARGE')
  HEADING CXTOTIO (I/O' 'CHARGE')
  HEADING CXTPTGCST ('TGET/TPUT' 'CHARGE')
  HEADING DEBIT ('TOTAL' 'CHARGE')
  LINE 01 CMJOBNM CMDTSTT CMJOBNO CXCONCHG +
          CXCPUCST CXTOTIO DEBIT
```

# Graphing Facility

---

This chapter describes the CA-Easytrieve Plus Toolkit graphing facility.

## Using the Facility

The CA-Easytrieve Plus Toolkit (Toolkit) graphing facility lets you produce graphic representation of data in your files. A graph is useful in determining relationships or trends in data and as a visual aid for data presentations in reports, analyses, and briefings.

The graphing facility produces four types of graphs:

- [Bar Graph](#)
- [Histogram](#)
- [Plot](#)
- [Deviation Bar Graph](#)

The graphing facility:

- Is designed primarily for use as a stand-alone program, graphing values from an input file
- Uses any file that can be input to Toolkit
- Operates by means of the macro invocation of Toolkit statements
- Uses freeform, nonpositional English-like keywords to control the graph format
- Produces all graphs on a standard line printer, eliminating the need for specialized output equipment

## Coding

The coding required to execute the graphing facility contains three essential parts:

- The Library section that describes your input file
- The FILE statement that describes the keyword file
- Statements that invoke the graphing facility

The required statements are discussed in [Restrictions](#) and [Required Statements](#). Sample JCL is illustrated later in [Example OS/390 and z/OS JCL](#).

## Restrictions

You cannot use the graphing facility with other Toolkit routines. You can use user-defined logic (for example, screening of input data) only if certain rules are observed. These rules are fully explained later in [Required Statements](#).

The subroutines used by the graphing facility are not reentrant. This means that multiple graphing jobs cannot be used in the same JCL job step. You must run each job individually.

Minimum and maximum values permitted as input to any of the graph routines are:

Minimum: -21,474,836.48  
Maximum: +21,474,836.47

If you input a value outside this range, unpredictable results occur.

The graphing facility requires 450 KB of storage for execution. Also, for DOS execution, the EXITSTR parameter in the environment section of a CA-Easytrieve Plus program must be specified as follows:

```
PARM EXITSTR 180
```

See the CA-Easytrieve Plus *Reference Guide* for an explanation of the EXITSTR parameter.

The graphing routines cannot specify a database file as the input file.

## Required Statements

The elements required to execute the graphing facility consist of the following major parts:

- [FILE Statements](#):
  - Input file statement
  - Keyword file statement
- [Graphing Facility Invocation Statement](#)
- An END statement
- File containing keyword commands

## FILE Statements

The two kinds of file statements are input and keyword.

### Input FILE Statement

The graphing facility accepts as input, any file that can be read by Toolkit. The input FILE statement is used to define this file.

### Keyword FILE Statement

Keywords are the commands you use to create customized graphic output. They are English-like terms (such as COLS, WIDTH, and ROWS) that specify the physical form of the graphs. You code these keywords in a file that you define and place within the job stream. The keywords to be coded are described individually for each graph type.

You define the keyword file by a FILE statement coded with the CARD parameter. The file name for the keyword file can be any combination of characters valid to Toolkit.

## Graphing Facility Invocation Statement

The generalized syntax for the graphing facility invocation statement follows:

### Syntax

```
%GRAPH1 infile type parmfile parms control-field f1 f2 f3 f4 f5 f6 sortfield  
%GRAPH2
```

infile

The name of the file that supplies input to GRAPH1 activity. A valid name can be any previously defined file.

type

The type of graph to be produced. Valid values are BAR, HIST, PLOT, or DEV.

parmfile

The name of the input file that contains the graphing facility keyword parameters. See [Operations](#) for additional information about parmfile.

parms

The 80-byte field in the keyword file that contains the keyword statements. In the example shown in parmfile, the value for parms is PRM.

control-field

The name of the nonquantitative alphanumeric field that contains the control break information for each graph (such as REGION or DEPT). You can name only one control field for each execution of the graphing facility.

f1, f2, f3, f4, f5, f6

The fields f1...f6 are used as input to the various graphing facility routines. Each routine uses these fields differently. All six parameters must be represented in the %GRAPH1 statement by an alphanumeric string separated by blanks.

**Note:** Any unused parameters must be coded with a zero.

sortfield

The name of the field used for sorting the input file. This parameter has restrictions which are described for each graph type.

%GRAPH2

Ending statement for the %GRAPH1 activity.

## Operations

Parmfile is defined using the CARD parameter in the FILE statement as shown in the CA-Easytrieve Plus *Reference Guide*. The format is:

```
FILE INCRD CARD  
PRM 1 80 A
```

## User Defined Screening Code

If records from the input file will be bypassed or otherwise altered before being input to the graphing facility, put the associated CA-Easytrieve Plus logic between the two graphing invocation statements (%GRAPH1 and %GRAPH2). If this code is placed anywhere else, syntax errors are generated. Input data screening for the graphing facility is identical to screening procedures for other routines.

## Example

The following generalized example illustrates the sequence in which the required statements must appear in the graphing facility coding. Complete JCL is provided later in this chapter.

The graphing invocation statements, introduced by %GRAPH1, must be followed by an ending statement, %GRAPH2, followed in turn by an END statement and the file containing the keywords.

Input FILE Statement	FILE MYFILE SALESYTD 1 8 N 2 AMTDUE 32 8 N 2 REGION 49 4 A
Keyword FILE Statement	FILE INCRD CARD PRM 1 80 A
Graphing Invocation Statements	%GRAPH1 MYFILE BAR INCRD PRM REGION SALESYTD AMTDUE 0 0 0 0 REGION %GRAPH2
END Card	END
File Containing Keyword Commands for Specific Graphs	1TITLE SAMPLE TITLE LINE FOR BAR GRAPH LINE 150000.00 FLD1,VAR,TOT FLD2,VAR,TOT BARKEY1 SALESYTD TOT BARKEY2 AMTDUE TOT ROWS 40 COLS 99 WIDTH 5 SPACE 2 BARS 1 1FOOT TEST FOOTER FOR BAR GRAPH

## Bar Graph

The standard bar graph facility produces a multiple-bar graphic that can use up to six variables (f1 through f6). The variables can be specified as fields from the input file, or entered as a constant, such as Sales-Target and Year-to-Date.

The same field can be used more than once on the same bar graph. This allows you to graph multiple values (minimum, maximum, mean, and so on) of a specific field, using one graph for all values instead of one graph for each value.

### Syntax

```
%GRAPH1 infile BAR parmfile parms control-field f1 f2 f3 f4 f5 f6 sortfield  
%GRAPH2
```

fn

You can use all six of the f1...f6 parameters. Each parameter used must have an associated FLDn keyword in your keyword file. The BARS keywords must indicate the number of f1...f6 parameters being used. This results in one bar being graphed for each f1...f6 parameter specified. Where desired, quoted literals can be used in place of field names. Code all unused parameters with zeroes.

sortfield/control-field

Sortfield must be the same field as control-field.

### Operation

Values for each variable entered in the routine are accumulated until the value of the control field changes. At that time a control break occurs, and a group of bars is produced, one for each variable. This process is repeated for each value of the control field until an end-of-file condition is reached on the input file.

Each bar on the printed output consists of a different special character that is supplied by the routine. This allows you to differentiate among variables. A legend can be printed at the top of the graph identifying the variable represented by each character or bar.

Multiple graphs are produced when the output exceeds the space of one graph. When this occurs, the graph continues on another output page.

## Keywords

The following keywords are used to format the Bar Graph. Coding examples follow the keyword explanations.

1TITLE  
2TITLE  
3TITLE

Three optional nTITLE lines are allowed, each of which can contain up to 40 characters of titling information. Any or all lines can be omitted. The nTITLE line sequence numbers are required. Each nTITLE line results in one title line being printed at the top of the graph.

Format:

```
1TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Output report title lines are blank if no titling information is entered.

ROWS

Specifies the number of horizontal rows to be used per graph. 1 is the minimum. 40 is the maximum and the default.

Format:

```
ROWS nn
```

COLS

Specifies the number of columns to be used as the total width of the graph. If the number is less than the total combined width of the graph desired (based on the number and width of the bars requested), the system uses the output default value. COLS nn includes both bars and spaces. 32 is the minimum. 99 is the maximum. 32 is the default.

Format:

```
COLS nn
```

LINE

When you specify the LINE parameter, the bar graph prints an additional value that is represented on the graph by a broken double line printed horizontally across the output report. This line is equivalent to the value's placement within the graph and results in a clear, visual comparison of the LINE value with the other data being graphed. A valid value for the LINE parameter is an actual numeric value between -2,147,483,648 and +2,147,483,647.

For example, LINE 500000.00 results in a significant value of 500,000.00 being displayed as a line across the graph. It is not necessary to code a decimal point when using whole numbers as significant values. (This could have been coded LINE 500000.)

To produce a mean (or average) line of the values graphed, code an M after the LINE keyword. This automatically calculates and prints the mean value of the input file.

If the value of the significance line is out of range of the file being graphed, the line appears on a level with the highest value reported.

Format:

LINE value M

No line shown is the default.

1FOOT  
2FOOT

You can specify two optional nFOOT lines, each of which can contain up to 40 characters of alphanumeric information. These lines appear on the last page of the output and are printed under the graph. The nFOOT sequence number must be given. Blank lines is the default.

Format:

1FOOT xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
2FOOT xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

WIDTH

Specify the width, in characters, of each bar. The minimum is 1 and there is no maximum. However, the maximum is limited by the number of bars desired, the space between each bar, and the width of the final printed output. If the bar's width causes it to compromise any of these factors, the default value is used. The default is 5.

If the total width of the graph (using the COLS keyword) is less than the total combined width of all bars, the default value of 5 is chosen by the routine. This allows a maximum of six bars to fit within the minimum 32 columns available for reporting.

Format:

WIDTH nn

## SPACE

Specify the number of blank spaces to appear between each bar group at control breaks. The minimum is 1, and there is no maximum. However, the maximum is limited by the number of bars desired, the width of each bar, and the width of the final printed output. If the number of spaces requested causes it to compromise any of these factors, the default value is used by the routine. The default is 1.

Format:

SPACE nn

## BARS

Establishes the number of bars to be graphed in each control group. The value you code must correspond to the number of non-zero Fn parameters specified in %GRAPH1.

**Note:** This keyword is required; the routine will not execute if this parameter is omitted.

The minimum is 1, and the maximum is 6. If more than six bars are requested, BARS defaults to 1.

Format:

BARS n

## FLDn , PARM1 , PARM2

Determines the method used to compute output values. By using different parameters, described next, you can control whether each bar represents a constant value or a variable, and in the case of a variable, whether the mean, minimum, maximum, count, or total value of the variable is graphed.

Format:

FLDn , parm1 , parm2

Where n is the sequence number of the bar being graphed. Commas are required between FLDn and parm1, and between parm1 and parm2.

**parm1** – Set to CON (CONstant) or VAR (VARiable).

The CON option, which does not have an associated parm2 value, results in the named field being used as a constant for reporting purposes. It is recommended that you place the constant value in the invocation of GRAPH1. (See [Example One](#)). Because no arithmetic calculations are performed on this field, the value of the field, at a control break, is used for the final graphic output.

The VAR option is used with one of the parm2 options described next. All values resulting from the use of the VAR option are taken at the control break and are reported for each control field.

**parm2**— Can be one of the following:

TOT: Causes the bar graph routine to calculate and display the TOTAl amount of the specified field for each control break.

MEAN: Causes the routine to calculate and display the MEAN (or average) value of all input records within each control break.

MAX: Graphs the MAXimum value found on the file within each control break.

MIN: Graphs the MINimum value found on the file within each control break.

CNT: Calculates and displays the number of occurrences of a field within control breaks. Generally, CNT is used with only one field (bar) because it gives a total of the number of records occurring within each control break.

For example, if there are 51 records for Region 101, with REGION being the control field, CNT will show a frequency of 51.

#### CONKEY

This keyword can contain up to 12 alphanumeric characters. It is used as an informational field by which you describe the control field being used (for example, REGION1).

When CONKEY is omitted, the control value field on the output is blank.

Format:

CONKEY xxxxxxxxxxxx

#### BARKEYn

This keyword can contain up to 12 alphanumeric characters. It is used as a descriptor field, allowing you to explain the keys, or special characters, used to display each bar. This field is displayed as a legend on the printed output. You can specify from one to six BARKEYn keywords.

When BARKEYn is omitted, the legend on the printed output is blank.

Format:

BARKEYn xxxxxxxxxxxx

Where n is the sequence number, from 1 to 6, of the bar being graphed.

## Examples

The following are two examples of the bar graph. Each example consists of the input, the bar graph produced, and an explanation of the graph.

Example One

Input

```

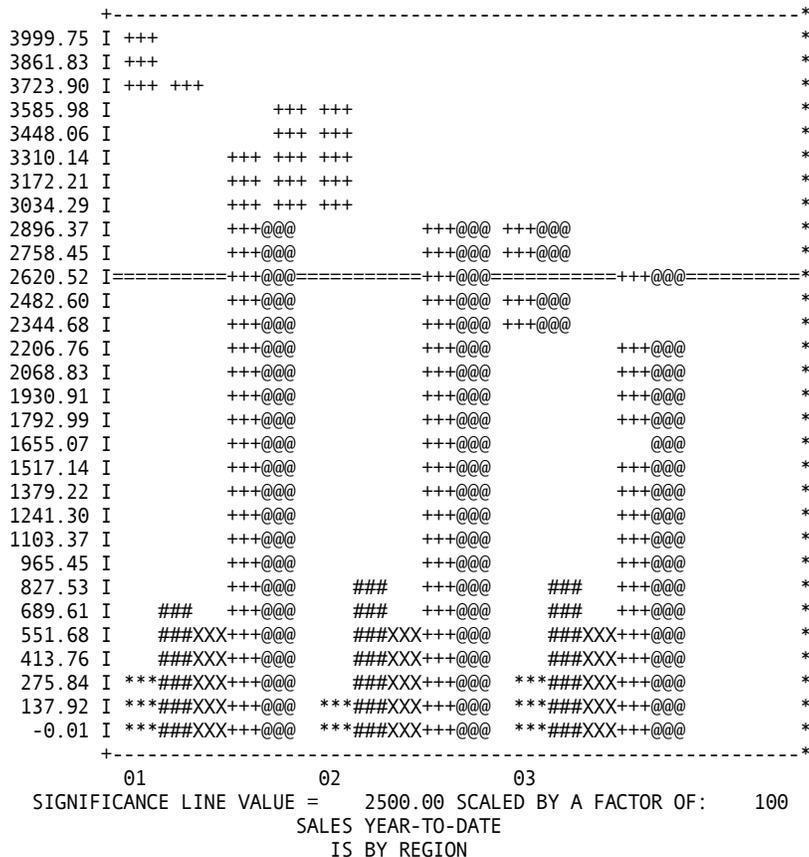
FILE INFILE ...
REGION 1 2 N
SD 3 8 P 2
FILE INCRD CARD
PRM 1 80 A
...
%GRAPH1 INFILE BAR INCRD PRM REGION SD SD SD SD 300000 0 REGION
%GRAPH2
END
    
```

Output

SAMPLE FOR BAR GRAPH EXAMPLE 1  
 ALL VALUES ARE FOR SALES-YEAR-TO-DATE  
 SHOWING MIN, MAX, MEAN, TOT AND CONSTANT

\* REPRESENTS SALESYTD MIN  
 # REPRESENTS SALESYTD MAX  
 X REPRESENTS SALESYTDMEAN  
 + REPRESENTS SALESYTD TOT  
 @ REPRESENTS \$300,000.00

ALL VALUES HAVE BEEN SCALED BY A FACTOR OF 100  
 CONTROL VALUE: REGION



Each set of bar graphs on the output is separated by two spaces (the SPACE 2 keyword). The control field chosen is REGION, which is represented as 01, 02, and 03 on the graph.

To keep the graph as simple as possible, quantitative values are factored, so values printed along the y axis do not become too large. The previous example is factored by 100. This function is performed automatically, and the factor varies depending on the size of the input values.

The increment value of the y axis is determined automatically by dividing the largest value to be graphed by the number of ROWS requested minus one. This creates a y axis with the specified number of rows ranging from 0 to the largest graphed value.

The significance line is printed at the value 2,500.00 (factored by 100) as specified by LINE 250000.00. Because no specific interval of 2,500.00 exists on the graph, the line is printed at the interval immediately above the true value of 2,500.00. The significance line in the bar graph does not overwrite the data.

On the box that outlines the graph, the right side consists of a column of asterisks (\*). This indicates that the graph shown is continued to another page of output. The graph on the next page of output is not shown but would contain a row of asterisks in the left-hand column to indicate that it is a continuation of a previous page of output. This continues until the final page is reached. The right side of the graph will consist of the character I. The bar graph in Example Two shows a graph indicating that it is the last page of the report.

## Example Two

Input to Example Two is shown next, followed by its output and an explanation of the graph.

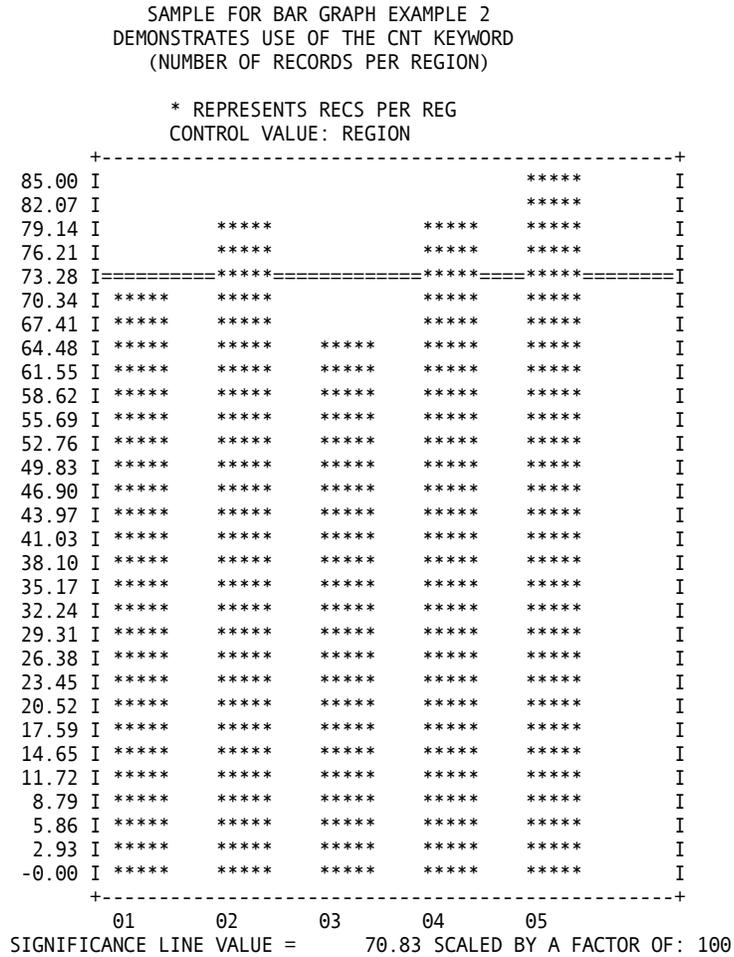
Input

```

FILE INFILE
REGION 1 2 N
SALESYTD 3 8 P 2
FILE INCRD CARD
PRM 1 80 A
%GRAPH1 INFILE BAR INCRD PRM REGION SALESYTD 0 0 0 0 REGION
%GRAPH2
END
1TITLE          SAMPLE FOR BAR GRAPH EXAMPLE 2
2TITLE          DEMONSTRATES USE OF THE CNT KEYWORD
3TITLE          (NUMBER OF RECORDS PER REGION)
LINE M
FLD1,VAR,CNT
BARKEY1 RECS PER REG
CONKEY REGION
ROWS 30
COLS 50
WIDTH 5
SPACE 4
BARS 1

```

Output



Because only one bar was specified (BARS 1), each bar on the graph represents one control break (in this case region). The significance line (70.83) is the average of the values (LINE M). No specific interval of 70.83 exists, so this line is placed at the value immediately above the true value of 70.83.

The values on the y axis are not scaled because the largest value is 85.00, so the scaling factor is 1, and the values printed on the y axis are the actual values. The right-hand line defining the graph box consists of the character I, indicating the end of the graph output.

## Histogram

The histogram produces a graph of the distribution of values on a y axis within intervals of x. The distribution shown can represent total, mean value, or frequency.

The x values are subdivided into intervals represented by upper and lower ranges. The value of y within each range of x values is represented by a bar on the output graph.

### Syntax

```
%GRAPH1 infile HIST parmfile parms control-field f1 f2 f3 f4 f5 f6 sortfield  
%GRAPH2
```

fn

The histogram uses only the f1 and f2 parameters. You must code all others with a zero. The f1 parameter gives the value of x, the f2 parameter the value of y.

sortfield

The sort field must be the same as the field you specify for the x value.

### Operation

The f1 parameter defines the x value desired, and the f2 parameter defines the y value. You cannot use literals with the histogram routine.

### Keywords

The following keywords are used to format the histogram. A coding example follows the keyword explanations.

```
1TITLE  
2TITLE  
3TITLE
```

Three optional nTITLE lines are allowed, each of which can contain up to 40 characters of titling information. Any or all lines can be omitted. The nTITLE line sequence numbers are required. Each nTITLE line will result in one title line being printed on the graph.

Output report title lines are blank if no titling information is entered.

Format:

```
1TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

ROWS

Specify the number of horizontal rows to be used per graph. The minimum is 1, and the maximum is 40. The default is 40.

Format:

```
ROWS nn
```

COLS

Specify the number of columns to be used as the total width of the graph. If the number is less than the total combined width of the graph desired (based on the number and width of the bars requested), the system uses the output default value.

The minimum is 32. The maximum is 99. The default is 32.

Format:

```
COLS nn
```

LINE

When you specify the LINE parameter, the bar graph prints an additional value that is represented on the graph by a broken double line printed horizontally across the output report. This line is equivalent to the value's placement within the graph, and results in a clear, visual comparison of the LINE value with the other data being graphed. A valid value for the LINE parameter is an actual numeric value between -2,147,483,648 and +2,147,483,647.

For example, LINE 150000.00 results in a significant value of 150,000.00 displayed as a line across the graph. It is not necessary to code a decimal point when using whole numbers as significant values. (This could have been coded LINE 150000.)

To produce a mean (or average) line of the values graphed, code an M after the LINE keyword. This automatically calculates and prints the mean value of the input file.

If the value of the significance line is out of range of the file being graphed, the line appears on a level with the highest value reported.

No line shown is the default.

Format:

LINE value

1FOOT  
2FOOT

You can specify two optional nFOOT lines, each of which can contain up to 40 characters of alphanumeric information. These lines appear on the last page of output and are printed under the graph. The nFOOT sequence number must be given.

The default is blank lines.

Format:

1FOOT xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
2FOOT xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

WIDTH

Specify the width in characters of each bar. The minimum is 1. The maximum is 10. The default is 5.

**Note:** If the total width of the graph desired (using the COLS keyword) is less than the total combined width of all bars, the default of 5 is used by the routine. This allows a maximum of six bars to fit within the minimum 32 columns available for reporting.

Format:

WIDTH nn

SPACE

Specify the number of blank spaces to appear between each bar group at control breaks. The minimum and default are 1. The maximum is 10.

Format:

SPACE nn

XDEF  
YDEF

These keywords can contain up to 12 alphanumeric characters each. They act as descriptor fields you can use to explain which values or fields from a file are used as the x and y values, respectively.

When these keywords are omitted, the XDEF and YDEF fields on the output are blank.

Format:

```
XDEF xxxxxxxxxxxx
YDEF xxxxxxxxxxxx
```

HISTYP

Controls the type of histogram printed.

Format:

```
HISTYP,parm1
```

**Note:** A comma is required between the keyword HISTYP and associated parm1 value. For example, HISTYP,MEAN.

By specifying one of the parm1 values described next, you can determine whether the total, mean, or count of y is represented for each interval.

- TOT: This parameter causes the routine to calculate and display the total y value within an x interval.
- MEAN: This parameter causes the routine to calculate the mean of y within a given interval.
- CNT: Gives a frequency of y within each interval.

Parm1 defaults to CNT.

INTVLS

Determines the number of intervals reported for each histogram. The intervals are determined using the routine by dividing the range of the x values in the file by the value specified for INTVLS.

The minimum is 1, and the maximum is 1000. The default is 10.

Format:

```
INTVLS nn
```

## Example

The following exhibits show an example of the histogram.

Input

```
FILE INFILE
DEPT 1 2 N
RATE 26 4 N 2
LOANAMT 31 14 N 2
FILE INCRD CARD
PRM 1 80 A
%GRAPH1 INFILE HIST INCRD PRM DEPT LOANAMT RATE 0 0 0 0 LOANAMT
%GRAPH2
```



This example demonstrates the HISTYP,CNT option of the histogram graph. The intervals on the x axis are automatically calculated by dividing the total range of x values by the requested number of intervals (INTVLS).

The increment value of the y axis is determined by dividing the largest value to be graphed by the number of ROWS requested minus one. This creates a y axis with the specified number of rows ranging from 0 to the largest graphed value.

The significance line, indicated by the horizontal line of the character =, occurs at the average of all graphed values. This was requested by the keyword parameter LINE M.

The right side of the box defining the graph consists of the character I. This indicates that the graph is not continued on another page.

## Plot

The plot routine provides a two-axis grid chart that graphs up to four fields as x coordinates against a fifth field (the y coordinates on the input file). Variables are plotted on the grid according to the coordinates received in the form of (y,x1,x2,x3,x4). Each x variable is represented by a different special character.

## Syntax

```
%GRAPH1 infile PLOT parmfile parms control-field f1 f2 f3 f4 f5 f6 sortfield
%GRAPH2
```

fn

Plot uses the f1 through f5 parameters. Code F6 with zero. Do not use literals with the Plot routine.

sortfield

The sort field must be the same as that specified for the y value.

## Operation

The plot routine of the graphing facility uses the f1 parm to define the y value being plotted. The f2 through f5 parms are used to designate the fields to be plotted as x values. The y value should have an associated YDEF keyword and the x values should have associated XnDEF keywords in the keyword file.

## Keywords

The following keywords are used to format the plot graph. A coding example follows the keyword explanation.

1TITLE  
2TITLE  
3TITLE

Three optional nTITLE lines are allowed, each of which can contain up to 40 characters of titling information. Any or all lines can be omitted. The nTITLE line sequence numbers must be coded. Each nTITLE line results in one title line being printed on the graph.

Output report title lines are blank if no titling information is entered.

Format:

```
1TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
2TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
3TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

ROWS

Specify the number of horizontal rows to be used per graph.

The minimum and default are 15. The maximum is 40.

Format:

```
ROWS nn
```

COLS

Specify the number of columns to be used as the total width of the graph. If the number is less than the total combined width of the graph (based on the number and width of the bars requested), the system uses the output default value.

The minimum and default are 32. The maximum is 99.

Format:

```
COLS nn
```

1FOOT  
2FOOT

There are two optional nFOOT lines, each of which can contain up to 40 characters of alphanumeric information. These lines appear on the last page of output and are printed under the graph. The nFOOT sequence number must be coded.

Blank lines is the default.

Format:

```
1F00T xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2F00T xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

X1DEF  
X2DEF  
X3DEF  
X4DEF

These keywords can contain up to 12 alphanumeric characters each. They cause a description of the specified x variables to be displayed on the printed output. One XnDEF keyword can be specified for each x variable reported.

No x value descriptor items are printed by default. The XnDEF fields on the output are blank.

Format:

```
X1DEF xxxxxxxxxxxx
X2DEF xxxxxxxxxxxx
X3DEF xxxxxxxxxxxx
X4DEF xxxxxxxxxxxx
```

YDEF

This keyword can contain up to 12 alphanumeric characters. It causes a description of the specified y variable to be displayed on the printed output.

No y value descriptor items are printed by default. The YDEF field on the output is blank.

Format:

```
YDEF xxxxxxxxxxxx
```

VARNUM

Specify the number of x variables being used. The minimum is 1. The maximum is 4.

VARNUM is a required keyword. The plot routine does not execute if you omit VARNUM.

Format:

```
VARNUM n
```

The value of n must coincide with the number of XnDEF keywords used.

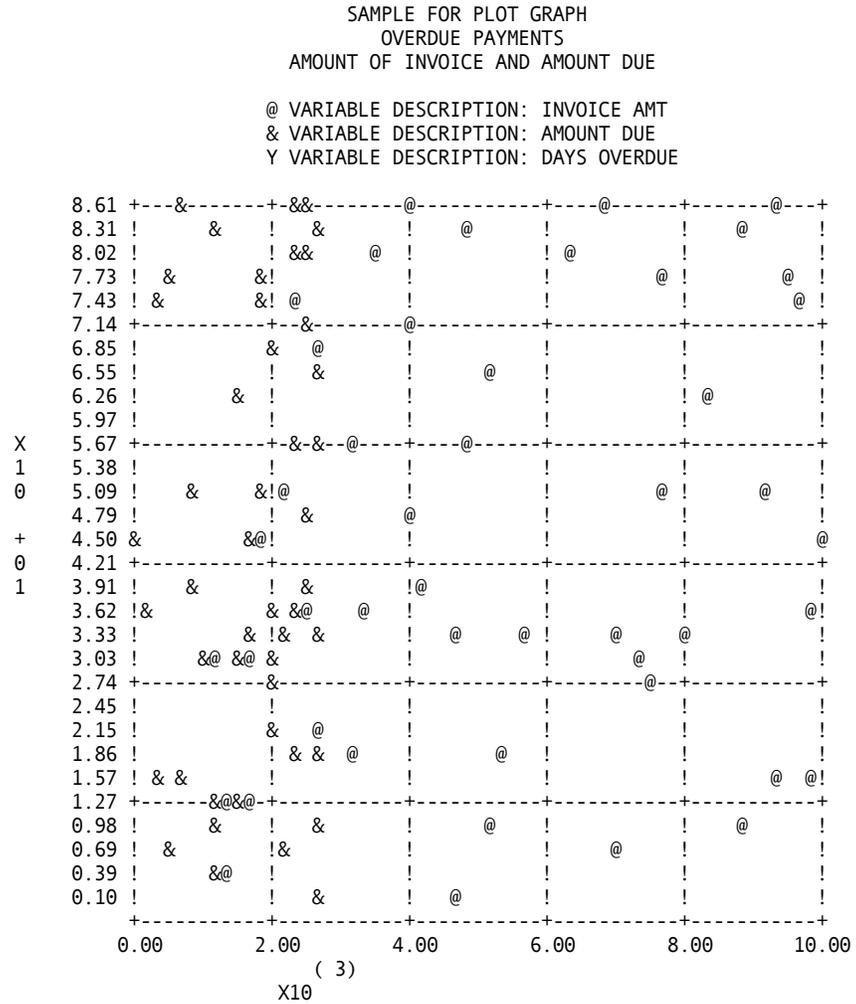
## Example

The following is an example of a plot graph.

Input

```
FILE INFILE
REGION 1 4 N 0
AMTINV 5 8 P 2
AMTDUE 21 8 P 2
OVERDUE 29 4 N 0
FILE INCRD CARD
PRM 1 80 A
%GRAPH1 INFILE PLOT INCRD PRM REGION OVERDUE AMTINV AMTDUE 0 0 0 OVERDUE
%GRAPH2
END
1TITILE          SAMPLE FOR PLOT GRAPH
2TITILE          OVERDUE PAYMENTS
3TITILE          AMOUNT OF INVOICE AND AMOUNT DUE
ROWS 30
COLS 60
YDEF DAYS OVERDUE
X1DEF INVOICE AMT
X2DEF AMOUNT DUE
VARNUM 2
```

Output



The increment values of both the x and y axes are calculated by dividing the largest value to be graphed by the value specified for COLS minus one and ROWS minus one, respectively. This creates an x and y axis with the specified number of columns and rows, and with values ranging from 0 to the largest value graphed.

In order to keep the graph as simple as possible, quantitative values are factored. In the previous example, the y axis is labeled as X 10 +01, which indicates that the values displayed should be multiplied by 10, and the x axis is labeled as X 10 (03), which indicates that the values should be multiplied by 1000.

## Deviation Bar Graph

The deviation bar graph supplies a graphic representation of the difference between the sums of two variables within control breaks. You can express this deviation as the numeric difference or as a percentage of difference.

### Syntax

```
%GRAPH1 infile DEV parmfile parms control-field f1 f2 f3 f4 f5 f6 sortfield  
%GRAPH2
```

fn

The Deviation routine uses only the f1 and f2 parameters. You must code all others as zero. The f1 and f2 values are the fields being used as the x1 and x2 values in the deviation calculation. Do not use literals with the deviation routine.

sortfield

The sort field must be the same field named as the control field.

### Operation

Values for each variable are accumulated until the value of the control field changes. At that time a control break occurs, and a horizontal bar is produced. This process is repeated for each value of the control field until end of file on the input file is reached.

On the deviation bar graph output, the control variable is listed as the left-hand scale. If, at a control break, the x2 variable is greater than the x1 variable, the horizontal bar representing their difference is on the negative side of the scale. If x2 is less than x1, the bar is graphed on the positive side.

The number of deviations that are graphed is determined by the number of control breaks encountered in the file. A maximum of 120 deviations is allowed. If more than 120 deviations are taken, an error message is generated, and processing stops.

### Keywords

The following keywords are used to format the deviation bar graph. A coding example follows the keyword explanation.

1TITLE  
2TITLE  
3TITLE

Three optional nTITLE lines are allowed, each of which can contain up to 40 characters of titling information. Any or all lines can be omitted. The nTITLE line sequence numbers must be coded. Each nTITLE line results in one title line being printed on the graph.

Output report title lines are blank if no titling information is provided.

Format:

```
1TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
3TITLE xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

COLS

Specify the number of columns that will be reported on each half of the deviation bar graph.

The minimum is 1. The maximum and default are 40.

Format:

```
COLS nn
```

LINE

When you specify the LINE parameter, the graph prints an additional value, represented on the graph by a column of special characters (! and #), that bisects the graph at the location corresponding to its value within the deviation. This allows you to clearly see how your file deviates from a significant value. A valid value for the LINE parameter is an actual numeric value between -2,147,483,648 and 2,147,483,647.

For example, LINE 500000.00 results in a significant value of 500,000.00 being displayed as a vertical line on the graph. It is not necessary to code a decimal point when using whole numbers as significant values. (This example could have been coded LINE 500000.)

To produce a mean (or average) line of the values graphed, code an M after the LINE keyword. This automatically calculates and prints the mean value of the input file.

If the value of the significance line is out of range of the file being graphed, no significance value is reported.

No line shown is the default.

Format:

LINE value

1FOOT  
2FOOT

You can specify two optional nFOOT lines, each of which can contain up to 40 characters of alphanumeric information. These lines appear on the last page of output and are printed under the graph. The nFOOT sequence number must be coded.

Blank lines is the default.

Format:

1FOOT xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
2FOOT xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

YVARS

Format:

YVARS xxxxxxxx

This optional field describes the control field being used. It can contain up to eight alphanumeric characters.

By default, no control value description is printed. When this keyword is omitted, the control value field on the printed graphic output is blank.

PCTDEV

Use this parameter to specify which type of deviation is graphed. 0 is the default.

Format:

PCTDEV n

There are two options for n:

- A zero indicates that the difference will be graphed.
- A one indicates that the difference as a percentage is graphed.

Example

This is an example of the deviation bar graph.

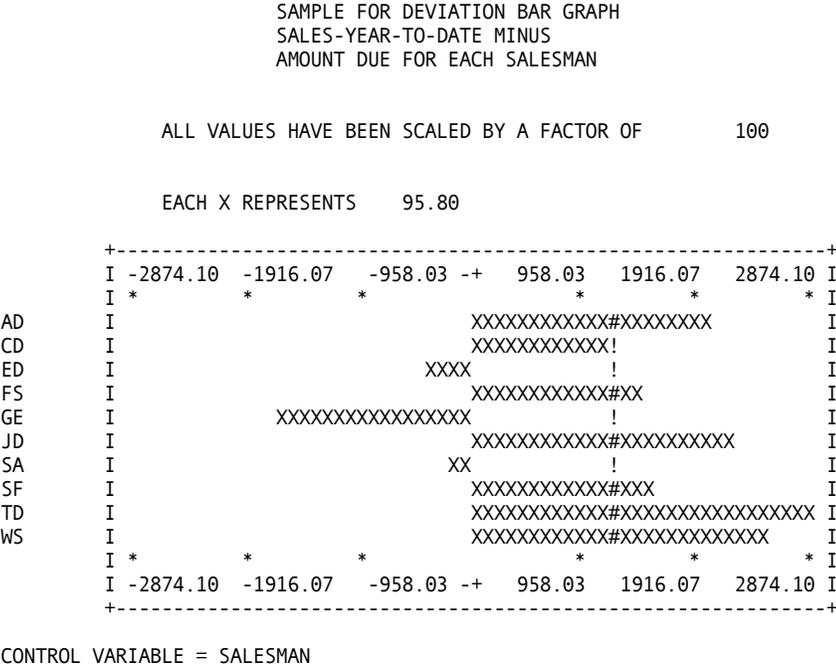
Input

```

FILE INFILE
SALESMAN 1 3 A
SALESYTD 4 8 P 2
AMTDUE 12 8 P 2
REGION 20 3 N
FILE INCRD CARD
PRM 1 80 A
%GRAPH1 INFILE DEV INCRD PRM SALESMAN SALESYTD AMTDUE 0 0 0 0 SALESMAN
%GRAPH2
END
1TITLE SAMPLE FOR DEVIATION BAR GRAPH
2TITLE SALES-YEAR-TO-DATE MINUS
3TITLE AMOUNT DUE FOR EACH SALESMAN
LINE M
COLS 30
YVARS SALESMAN
PCTDEV 0

```

Output



This example demonstrates the difference option of the deviation bar graph, which is specified by the PCTDEV 0 keyword. For each salesman listed on the y axis, the value for amount due is subtracted from sales-year-to-date and is represented by a horizontal bar graph.

The significance line, indicated by the vertical line of the characters ! and #, occurs at the average of all graphed values. This was requested by the keyword parameter LINE M. The character ! is printed as the significance line when the position is blank. When the position contains the character X, the character # is printed for the significance line.

To keep the graph as simple as possible, quantitative values are factored so that values printed along the x axis do not become too large. In the example, the values printed are factored by 100. This function is performed automatically, and the factor varies depending on the size of the input values.

## Example OS/390 and z/OS JCL

The JCL required to execute the graphing routines is the same as required to execute any Toolkit program, with the addition of the following DD statement:

```
//OUTPUT DD SYSOUT=A (data set used for output)
```

The exact format of this statement may vary slightly. The following example shows JCL, the required invocation statements, and appropriate keywords:

```
//jobname JOB accounting.info
//STEPNAME EXEC PGM=EZTPA00,REGION=512K
//STEPLIB DD DSN=your.caeztpls.loadlib,DISP=SHR
// DD DSN=your.capaupls.loadlib,DISP=SHR

//PANDD DD DSN=your.capaupls.macro.library,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSSNAP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//EZTVFM DD UNIT=SYSDA,SPACE=(CYL,(10,2))
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(15,5))
//SORTWK02 DD UNIT=SYSDA,SPACE=(CYL,(15,5))
//SORTWK03 DD UNIT=SYSDA,SPACE=(CYL,(15,5))
//INREC DD ...
//SYSIN DD *
FILE INREC
  SALESYTD 1 8 N 2
  COMMYTD 9 8 N 2
  REGION 49 4 A
FILE INCRD CARD
PRM 1 80 A
%GRAPH1 INREC BAR INCRD PRM REGION SALESYTD COMMYTD 0 0 0 0 REGION
%GRAPH2
END
1TITL SAMPLE TITLE LINES FOR
2TITL TEST OF THE STANDARD BAR GRAPH
LINE 150000
FLD1,VAR,MAX
FLD2,VAR,MEAN
BARKEY1 SALESYTD MAX
BARKEY2 COMMYTD MEAN
ROWS 40
COLS 99
WIDTH 10
SPACE 3
```

```
BARS 2  
1FOOT TEST FOOTER FOR SAME REASON  
/*  
//
```



# Advanced Techniques

---

Due to the macro-based design of CA-Easytrieve Plus Toolkit (Toolkit), you can design sophisticated auditing applications without learning complicated programming languages, syntax, or procedures.

However, because of the advanced power of the host language, CA-Easytrieve Plus, and the flexibility of the design of Toolkit routines, you can create complex applications. This chapter describes some of the advanced techniques that you can use with Toolkit routines. These advanced techniques are based on the features of CA-Easytrieve Plus.

This chapter covers the following advanced techniques:

- Use of CA-Easytrieve Plus procedures
- Stand-alone DISPLAY and stand-alone REPORT routines
- Logic-after-invocation section of stand-alone routines
- Logic-before-invocation section of stand-alone routines
- Use of sample flags
- Stacking stand-alone routines
- Database use of Toolkit
- Miscellaneous techniques

The end of this chapter contains a functional chart of inline and stand-alone routines and a summary of the advanced techniques.

## CA-Easytrieve Plus Procedures

Toolkit routines are written in CA-Easytrieve Plus, an information retrieval and data management system. One of the features of CA-Easytrieve Plus is the use of procedures for defining your own routines within a program. By defining your own routines within a program, logic flow is simplified.

Define procedures within a program with the PROC and END-PROC keywords and invoke procedures with the PERFORM statement (see the CA-Easytrieve Plus *Reference Guide* for details). For example, consider the following code which demonstrates the use of procedures to simplify logic flow:

```
...
JOB INPUT ...
PERFORM TASK1
PERFORM TASK2
PERFORM TASK1
PERFORM TASK3
PERFORM TASK1
PERFORM TASK4
*
TASK1. PROC
.
.
.
END-PROC
*
TASK2. PROC
.
.
.
END-PROC
*
TASK3. PROC
.
.
.
END-PROC
*
TASK4. PROC
.
.
.
END-PROC
```

In this example, the PROC and END-PROC statements are used to define four procedures. The PERFORM statement invokes these procedures. Performing these procedures eliminates the need for repeating sections of code, branching statements, or other complex logic that inline coding techniques require. Instead, the procedures define routines within the main routine. The PERFORM statement can invoke these procedures any time.

Stand-alone Toolkit routines extensively use this method of coding, called structured coding. You should use structured coding techniques although the same results can be obtained with inline techniques. This is because programs are easier to write and debug when they use structured techniques.

Procedures are an integral part of structured coding, and you can use them in many areas of Toolkit, such as:

- With the logic-before-invocation section of some stand-alone routines
- With the logic-after-invocation section of some stand-alone routines
- With inline routines
- With REPORT subactivities
- With the sampling flags technique

The next example demonstrates the use of procedures in a Toolkit routine.

### Example - Procedure with an Inline Routine

```

FILE PAYFILE
  EMPLOYEE-NAME      1  20  A
  EMPLOYEE-CODE      21  1  A
  MEDICAL-DEDUCT     23  3  P  2
  SSN                 31  9  N
*
JOB INPUT PAYFILE
PERFORM SCREEN-FILE
PERFORM VALIDATE
*
SCREEN-FILE. PROC
IF EMPLOYEE-CODE EQ 'P'
  GO TO JOB
END-IF
END-PROC
*
VALIDATE. PROC
%NUMTEST MEDICAL-DEDUCT 'MEDICAL-DEDUCT NOT NUMERIC' +
  EMPLOYEE-NAME
IF NUMTEST-FLAG EQ 'YES' AND MEDICAL-DEDUCT NE 0
  PRINT VALID-MED
END-IF
END-PROC
*
REPORT VALID-MED
TITLE 01 'EMPLOYEES WITH MEDICAL DEDUCTIONS'
LINE EMPLOYEE-NAME SSN MEDICAL-DEDUCT

```

The code in this sample demonstrates how procedures are used with the inline routine, NUMTEST. The first portion of code is the library section. The next section of code is the JOB INPUT PAYFILE statement. It identifies the automatic input file and also contains two PERFORM statements which execute the procedures.

The SCREEN-FILE procedure executes the GO TO JOB statement if the employee is part time. The VALIDATE procedure checks for a valid numeric value in the MEDICAL-DEDUCT field. If the value is valid and nonzero, the VALIDATE procedure prints a line of the VALID-MED report.

Note the order of the various sections of code:

1. Library section
2. Mainline code – JOB INPUT and PERFORM statements
3. Definition of procedures
4. Definition of REPORT

For complete details regarding the correct order of statements, see the CA-Easytrieve Plus *Reference Guide*.

### Procedure Placement

Place CA-Easytrieve Plus procedures immediately after the associated activity (JOB or SORT) or subactivity (REPORT).

Place procedures that you use with stand-alone routines either after the Toolkit macro in the logic-after-invocation section or before the Toolkit macro in the logic-before-invocation section. Do not place procedures between the first and second macro invocations. For more details, see [Stand-alone Routines \(DISPLAY and REPORT\)](#).

## Stand-alone Routines (DISPLAY and REPORT)

Stand-alone DISPLAY and stand-alone REPORT routines are two distinct types of Toolkit routines. However, in most applications, the difference between these two types of stand-alone routines is transparent. The only time that their difference is important is when an application uses the logic-after-invocation section of stand-alone routines. These two subjects, stand-alone DISPLAY and stand-alone REPORT routines and logic-after-invocation in stand-alone routines, are related and are discussed together in this section.

### Logic-After-Invocation of Stand-alone Routines

Stand-alone DISPLAY and stand-alone REPORT routines are differentiated by the method that they use to create a listing. The stand-alone DISPLAY routine uses the CA-Easytrieve Plus DISPLAY statement, and the stand-alone REPORT routine uses a REPORT subactivity to create listings. Because of the different requirements of DISPLAY and REPORT statements, there is a corresponding difference in the use of the logic-after-invocation section for these routines.

The four types of CA-Easytrieve Plus statements that you can code in the logic-after-invocation section are:

- Procedures
- JOB or SORT activities
- REPORT statements
- Special-name REPORT procedures

### Stand-alone REPORT

Stand-alone REPORT routines cannot define a procedure in the logic-after-invocation section. This is because of the REPORT statement at the end of all stand-alone REPORT routines. A procedure must be placed immediately after the associated activity. Therefore, a procedure in the logic-after-invocation section of a stand-alone REPORT routine is associated with the REPORT subactivity, not the previous JOB or SORT activity. The only procedure that can follow a stand-alone REPORT routine is a special-name REPORT procedure.

Stand-alone REPORT routines cannot define a REPORT statement in the logic-after-invocation section. This is because of the structure used in stand-alone REPORT routines. Since most stand-alone REPORT routines use multiple CA-Easytrieve Plus activities, the screening code section, in which the report is printed, is associated with a different activity from where the REPORT statement is placed.

### Stand-alone DISPLAY

A stand-alone DISPLAY routine can use a procedure in the logic-after-invocation section because a stand-alone DISPLAY routine does not use a REPORT subactivity. The remaining two types of CA-Easytrieve Plus statements, which can be associated with both stand-alone DISPLAY and stand-alone REPORT routines, are JOB or SORT activities and special-name REPORT procedures.

### JOB or SORT Activities

A JOB or SORT activity defines a new activity that is not directly related to the Toolkit routine. This new activity can take the form of a user-coded activity or an activity encountered through the coding of an additional Toolkit routine (see [Stacking Stand-alone Routines](#) later in this chapter).

## Special-Name REPORT Procedures

You can also use the logic-after-invocation section with special-name REPORT procedures.

The REPORT subactivity has special-name REPORT procedures that you can define to perform various functions. When you code a special-name procedure, the logic within the procedure is performed at the appropriate time for its function (see the *CA-Easytrieve Plus Reference Guide*).

For example, when you code the ENDPAGE procedure, it executes every time Toolkit reaches the end of report body. You can use this procedure to produce footers at the bottom of each page of a report. For a list and description of these procedures, see the *CA-Easytrieve Plus Reference Guide*.

You can code special-name REPORT procedures in both types of stand-alone routines. Because stand-alone REPORT routines always end with a REPORT, code the special-name REPORT procedure immediately after the second macro invocation. Since stand-alone DISPLAY routines do not contain a REPORT, use stand-alone DISPLAY routines only if you define a REPORT in the logic-after-invocation section.

Some stand-alone REPORT routines use special-name REPORT procedures. When this occurs, you cannot define that special-name REPORT procedure in the logic-after-invocation section of the stand-alone REPORT routine. This is because special-name REPORT procedures can only be used once for a given report. The following lists these stand-alone REPORT routines and the special-name REPORT procedures that they use:

Routine	Special-Name REPORT Procedures Used
INTERVL	BEFORE-BREAK, TERMINATION
OCCURS	BEFORE-BREAK
STRATIF	BEFORE-BREAK, AFTER-BREAK
VERSUS	BEFORE-BREAK, TERMINATION

Additional examples of the use of procedures in Toolkit are shown in subsequent topics in this chapter.

## Placement of Procedures or Reports

The method that you use to code procedures and REPORTs in the logic-after-invocation section are identical. You perform a procedure or print a REPORT in the screening code section. Then, code the procedure that you want to perform, or the REPORT that you want to print, in the logic-after-invocation section of the stand-alone routine. If both procedures and REPORTs are used with the stand-alone DISPLAY routine, the statements defining the procedures must precede the REPORT statements. This is because the REPORT and any associated REPORT procedures must be the last statements in any CA-Easytrieve Plus job activity.

You must also code special-name REPORT procedures in the logic-after-invocation section. These special procedures must immediately follow the associated REPORT. If a program contains a procedure, REPORT, and a special-name REPORT procedure, the order of occurrence must be:

1. Procedure
2. REPORT
3. Special-name REPORT procedure

The special-name REPORT procedure is the only type of procedure that you can specify in the logic-after-invocation section of a stand-alone REPORT routine.

[Example Three - Special-Name REPORT Procedure](#) demonstrates this combination.

### Example One - Procedure in a Stand-alone DISPLAY Routine

```

FILE PAYFILE
  EMPLOYEE-CODE    5  1  A
  GROSS-PAY        23  5  P  2
  HIRE-DATE        31  5  N
FILE SAMPFIL
*
%DOLUNIT1 PAYFILE GROSS-PAY 10000 2000 13453
PERFORM SCREEN1
PERFORM SCREEN2
%DOLUNIT2 SAMPFIL
*
SCREEN1. PROC
IF EMPLOYEE-CODE EQ 'P'
  GO TO JOB
END-IF
END-PROC
*
SCREEN2. PROC
%DAYSAGOL HIRE-DATE YYDD LT 30
IF DAYSAGO-FLAG EQ 'YES'
  GO TO JOB
END-IF
END-PROC

```

Instead of coding logic in the screening code section, the example performs a procedure. The logic-after-invocation section defines the procedure, and the procedure performs the same screening function as the example in the chapter [“Coding Guidelines.”](#) The Logic-After-Invocation section can define a procedure because DOLUNIT is a stand-alone DISPLAY routine and does not use a REPORT subactivity.

The PERFORM SCREEN1 statement invokes the SCREEN1 procedure. The SCREEN1 procedure bypasses the processing of records having an EMPLOYEE-CODE of P. The PERFORM SCREEN2 statement invokes the SCREEN2 procedure. The SCREEN2 procedure invokes the inline routine DAYSAGOL to determine if the HIRE-DATE is less than 30 days old. If this is true, then the GO TO JOB statement bypasses the record.

### Example Two - Procedure and REPORT in a Stand-alone DISPLAY Routine

```

FILE PAYFILE
  EMPLOYEE-NAME    1 20  A
  GROSS-PAY        23 5  P 2
  SSN               31 9  N
FILE SAMPFIL
*
%DOLUNIT1 PAYFILE GROSS-PAY 10000 2000 13453
PERFORM CUTOFF-CHECK
%DOLUNIT2 SAMPFIL
*
CUTOFF-CHECK. PROC
IF GROSS-PAY GE 2000
  PRINT TOP-ITEMS
END-IF
END-PROC
*
REPORT TOP-ITEMS
SEQUENCE EMPLOYEE-NAME
TITLE 01 'EMPLOYEES WITH GROSS PAY EXCEEDING CUTOFF'
LINE EMPLOYEE-NAME SSN GROSS-PAY

```

Since DOLUNIT is a stand-alone DISPLAY routine, procedures can be coded in the logic-after-invocation section. In this example, the CUTOFF-CHECK procedure is performed in the screening code section. The CUTOFF-CHECK procedure checks to see if GROSS-PAY is greater than or equal to the cutoff value, 2000. If this condition is true, a line of the TOP-ITEMS REPORT is printed. Because both a procedure and a REPORT are used, the statements defining the procedure occur before the REPORT statements.

### Example Three - Special-Name REPORT Procedure

The following example demonstrates the use of a special-name REPORT procedure with a stand-alone REPORT routine:

```

FILE PERSNL FB(150 1800)
REGION 1 1 N
*
%OCCURS1 PERSNL GRAPH 0 2
%OCCURS2 REGION
*
ENDPAGE. PROC
  DISPLAY +20 'CONFIDENTIAL - FOR AUDITORS ONLY'
END-PROC

```

In this example the ENDPAGE REPORT procedure is used to display a footer at the bottom of the report. The REPORT subactivity automatically performs the ENDPAGE procedure when the bottom of the report body is reached. It is not necessary for you to perform any of the special-name REPORT procedures. They are performed at the appropriate time for their function as the CA-Easytrieve Plus *Reference Guide* describes.

## Logic-Before-Invocation of Stand-alone Routines

The logic-before-invocation section in stand-alone routines allows you to define CA-Easytrieve Plus activities prior to the invocation of the Toolkit routine. This allows you to define multiple activities in a single execution of Toolkit.

Statements that you code in the logic-before-invocation section must contain a complete CA-Easytrieve Plus activity (JOB or SORT) and can contain a REPORT subactivity. Common examples of activities that you can perform in the logic-before-invocation section are:

- Pre-calculation techniques
- Creating an input file from two or more files

The logic-before-invocation section is located after the library section and prior to the first macro invocation. This means that any input or output file for the logic-before-invocation section must be defined in the library section. (There can be only one library section in a single execution of Toolkit.) If you want to screen, reconstruct, or merge an input file or files in the logic-before-invocation section, you must create a new temporary or permanent file. Specify the temporary or permanent file as input to the Toolkit routine. The following examples demonstrate this technique.

### Limitation

Some stand-alone routines contain FILE statements that do not allow you to use the logic-before-invocation section. The beginning of the Toolkit routine (which follows the library section) contains these FILE statements. With these routines it is not possible to code a CA-Easytrieve Plus activity between the library section and macro invocation statement because the code would be placed in the middle of the library section.

The following is a list of routines that do not support the logic-before-invocation section:

ADDRCMP	MULTDUP
CAVEVAL	SRCECOMP
CBLCNVRT	STOPORGO
DUPTST	STRATIF
ENCRYPT	STRTEVL
INTERVL	VERSUS

The following examples demonstrate techniques for using the logic-before-invocation section.

### Example One - Pre-Calculation Technique

```
FILE INFILE ...
FILE SAMPFIL ...
*
```

```
%POPSIZE1 INFILE
%POPSIZE2
*
%RANDPCT1 INFILE POP-SIZE 2.0 984875
%RANDPCT2 SAMPFIL
```

The RANDPCT routine requires an exact count of the population size as a parameter. One method of obtaining this information is to run an INTERVL analysis, or another CA-Easytrieve Plus program, to obtain the record count. Then, put the value of the record count into the parameter list. This method is time-consuming and requires an extra execution of Toolkit to obtain the population size.

Example One demonstrates a pre-calculation technique that automatically supplies the population size to RANDPCT. The logic-before-invocation section consists of the Toolkit routine POPSIZE. POPSIZE finds the population of the file and assigns it to the field POP-SIZE. POP-SIZE is then specified as the size parameter in the RANDPCT routine.

The pre-calculation technique ensures that the exact population size is always specified in the appropriate sampling routines. Use this technique for all sampling routines that require an exact population size as a parameter.

### Example Two - Creating an Input File from Two or More Files

```
FILE REGSMP
  CUSTOMER-ID          1  6  N
  AMOUNT-DUE           27  6  P  2
FILE REGAUD
  AUD-CUST-ID          1  6  N
  AUD-AMT-DUE          7 10  N  2
FILE AUDSMP
  AUDSMP-CUST-ID       1  6  N
  REC-AMT              7  6  P  2
  AUD-AMT              13  6  P  2
DEFINE REPLIES-USED    S  5  P  0  VALUE (0)
DEFINE REPLIES-NOT-USED S  5  P  0  VALUE (0)
DEFINE STOP-FLAG       S  3  A    VALUE ('NO')
*
JOB INPUT (REGSMP KEY CUSTOMER-ID +
           REGAUD KEY AUD-CUST-ID) FINISH END-OF-JOB
IF NOT MATCHED AND REGAUD
  STOP-FLAG = 'YES'
  PRINT NO-MATCH
  GO TO JOB
END-IF
IF MATCHED
  AUD-AMT = AUD-AMT-DUE
  REPLIES-USED = REPLIES-USED + 1
END-IF
IF NOT MATCHED AND REGSMP
  AUD-AMT = AMOUNT-DUE
  REPLIES-NOT-USED = REPLIES-NOT-USED + 1
END-IF
REC-AMT = AMOUNT-DUE
AUDSMP-CUST-ID = CUSTOMER-ID
PUT AUDSMP
END-OF-JOB. PROC
```

```

PRINT REPLIES
IF STOP-FLAG EQ 'YES'
  STOP-EXECUTE
END-IF
END-PROC
REPORT NO-MATCH
TITLE 01 'NON-MATCHING CUSTOMER IDS'
LINE 01 AUD-CUST-ID
REPORT REPLIES
TITLE 01 'SUMMARY OF CUSTOMER REPLIES IN AUDSMP FILE'
LINE 01 REPLIES-USED REPLIES-NOT-USED
%REGSAM1 AUDSMP REC-AMT AUD-AMT 32555
%REGSAM2

```

Example Two demonstrates the creation of the input file to REGSAM from the data of two separate files. The REGSMP file is the preliminary sample file to REGSAM. The REGAUD file is created using information from customer replies to confirmation letters regarding the accuracy of the data in the preliminary sample file. After the execution of the code, the AUDSMP file contains the proper recorded and audited amounts obtained from the REGSMP and REGAUD files.

This job performs synchronized file processing. The JOB INPUT statement defines the two files to be processed and the keys that are to be matched (see the CA-Easytrieve Plus *Reference Guide*).

The first IF statement checks for a nonmatched condition with the extra record being in the REGAUD file. If this is true, a record exists on the audited file with no match on the sample file. This is an error condition and means that an incorrect customer ID was entered when the REGAUD file was created. In this case, the job sets the STOP-FLAG to 'YES' and prints the nonmatching customer ID. The GO TO JOB statement bypasses the processing of this record.

The next IF statement checks for a matched condition. If this is true, an audited AMOUNT-DUE was obtained from a customer. The job assigns this amount to the AUD-AMT field in the AUDSMP file. Also, the job increments a counter named REPLIES-USED to keep track of the number of customer replies in the AUDSMP file.

The final IF statement checks for a nonmatched condition with the extra record being in the REGSMP file. If this is true, a record exists on the sample file with no match on the audited file. This condition is acceptable and means that the customer did not reply to the confirmation letter with an audited amount. In this case, it is assumed that the audited amount is the same as the recorded amount, and the job makes the appropriate assignment. Also, the job increments a counter named REPLIES-NOT-USED to keep track of the number of nonreplies in the AUDSMP file. Then, the job assigns the AMOUNT-DUE field to the REC-AMT field for processing by the subsequent REGSAM routine. The job then writes the record to the AUDSMP file.

When both files reach the end of file, the END-OF-JOB procedure is automatically performed. This procedure prints the summary of customer replies and checks the STOP-FLAG for a value of YES. If this condition is true, the previously described error condition was encountered, and the STOP-EXECUTE statement prevents the execution of REGSAM. Otherwise, the REGSAM routine is executed.

## Use of Sample Flags

Toolkit routines that create sample files use sample flags. Sample flags are reserved fields indicating whether a given record was selected for the sample file. If the reserved field contains the value YES, the record was selected for the sample file; if it contains NO, the record was not selected.

The PERFORM procname parameter identifies the routines that use the sample flag technique. The procname is a procedure that the routine performs after the value in the reserved field is set to YES or NO. This provides the opportunity to define a procedure in the logic-after-invocation section to print a report or perform other activities based on the selection of records for the sample population.

The following routines use the sample flag technique:

ATTSAMP	RANDPCT
CAVSAMP	RANDXCT
DISCSMP	REGSAMP
DOLUNIT	SPS
EACHNTH	STOPORGO
INTSAMP	VARSAAMP

The sample flag technique allows you to perform various functions for both positive and negative sampling. You can print listings of records selected or records not selected. It is also possible to create a file of records not selected or perform complex processing logic such as creating reformatted files.

Since you apply the sample flag technique by coding a procedure, there is great flexibility in the type of results that you can obtain. The following examples illustrate three different methods of using sample flags to create listings.

### Example One - Sample Flags with DISPLAY

Input

```
FILE PAYFILE ...
  NET          5   6   P  2
  EMPLOYEE-NAME 24  20  A
  ...
FILE SAMPFIL ...
*
%SPS1 PAYFILE NET 2000 93848
```

```

%SPS2 SAMPFIL PERFORM SELECT-LIST
SELECT-LIST. PROC
  IF SPS-SELECTED EQ 'YES'
    DISPLAY 'SELECTED' +2 EMPLOYEE-NAME
  END-IF
END-PROC

```

This example demonstrates a sampling proportional to size algorithm on a net pay field in a payroll file. The PERFORM SELECT-LIST parameter invokes the sample flag technique. This tells the SPS routine to set the reserved field, SPS-SELECTED, to the value YES or NO, depending on whether a given record is selected for the sample file. The routine then performs the SELECT-LIST procedure.

The user codes the SELECT-LIST procedure in the logic-after-invocation section of the program. This procedure tests the SPS-SELECTED field for the value YES. If this condition is true, then the word SELECTED and the employee's name are printed.

## Output

```

                                SPS SAMPLING REPORT
                                INPUT PARAMETERS

                                INPUT FILENAME                INFILE
                                INPUT FIELD                    NET
                                VALUE OF INPUT FIELD IS        ABS
                                TARGET VALUE                   20,000.00

                                SAMPLE FILE

SELECTED EMPLOYEE37
SELECTED EMPLOYEE76
SELECTED EMPLOYEE112
SELECTED EMPLOYEE146
SELECTED EMPLOYEE186
SELECTED EMPLOYEE224
SELECTED EMPLOYEE263
SELECTED EMPLOYEE301
SELECTED EMPLOYEE336
SELECTED EMPLOYEE376
SELECTED EMPLOYEE420
SELECTED EMPLOYEE456
SELECTED EMPLOYEE494
SELECTED EMPLOYEE535
SELECTED EMPLOYEE572
SELECTED EMPLOYEE606
SELECTED EMPLOYEE641
SELECTED EMPLOYEE682
SELECTED EMPLOYEE718
SELECTED EMPLOYEE755
SELECTED EMPLOYEE793
SELECTED EMPLOYEE833
SELECTED EMPLOYEE872
SELECTED EMPLOYEE912

                                NUMBER OF RECORDS PROCESSED      928
                                ABS VALUE OF RECORDS PROCESSED    488,886.12
                                ACT VALUE OF RECORDS PROCESSED    488,886.12
                                POS VALUE OF RECORDS PROCESSED    488,886.12
                                NUMBER OF RECORDS IN SAMPLE FILE   24

                                FILE SAMPFIL WILL BE CREATED

```

The output list shows the input parameters and the beginning of the sample file statistics. However, the list of selected employees is printed before the sample file results are printed. This is because the SELECT-LIST procedure contains a DISPLAY statement that immediately prints a line. These lines of output occur before the sample file statistics because the statistics are printed after the sampling of all records.

## Example Two - Sample Flags with REPORT

Input

```
FILE PAYFILE ...
  NET          5    6    P  2
  EMPLOYEE-NAME 24   20   A
  ...
FILE SAMPFIL ...
*
%SPS1 PAYFILE NET 2000 93848
%SPS2 SAMPFIL PERFORM SELECT-LIST
SELECT-LIST. PROC
  IF SPS-SELECTED EQ 'YES'
    PRINT SELECT-RPT
  END-IF
END-PROC
*
REPORT SELECT-RPT
TITLE 01 'SPS SELECTION LIST OF PAYFILE'
LINE EMPLOYEE-NAME NET
```

This example performs the same function as Example One, but instead of using a DISPLAY statement to produce the selection list, a REPORT statement is used. The PRINT SELECT-RPT statement replaces the DISPLAY statement, and a line of the report is generated each time the SPS-SELECTED field contains the value YES.

The report specifies a title and lists each employee's name and their associated net pay. The logic-after-invocation section contains both a procedure and a REPORT, with the procedure preceding the REPORT statements.

Output

```
          SPS SAMPLING REPORT
          INPUT PARAMETERS

INPUT FILENAME          INFILE
INPUT FIELD             NET
VALUE OF INPUT FIELD IS ABS
TARGET VALUE            20,000.00

          SAMPLE FILE
          SPS SELECTION LIST OF PAYFILE
                                     PAGE    1

          EMPLOYEE-NAME          NET
          EMPLOYEE37             165.34
          EMPLOYEE76             374.71
          EMPLOYEE112            952.28
          EMPLOYEE146            396.86
```

EMPLOYEE186	228.74
EMPLOYEE224	504.76
EMPLOYEE263	748.33
EMPLOYEE301	779.90
EMPLOYEE336	953.15
EMPLOYEE376	797.40
EMPLOYEE420	701.52
EMPLOYEE456	869.73
EMPLOYEE494	483.01
EMPLOYEE535	752.38
EMPLOYEE572	171.24
EMPLOYEE606	962.78
EMPLOYEE641	626.59
EMPLOYEE682	937.00
EMPLOYEE718	480.92
EMPLOYEE755	388.09
EMPLOYEE793	428.20
EMPLOYEE833	295.36
EMPLOYEE872	844.52
EMPLOYEE912	821.08

NUMBER OF RECORDS PROCESSED	928
ABS VALUE OF RECORDS PROCESSED	488,886.12
ACT VALUE OF RECORDS PROCESSED	488,886.12
POS VALUE OF RECORDS PROCESSED	488,886.12
NUMBER OF RECORDS IN SAMPLE FILE	24

FILE SAMPFIL WILL BE CREATED

This example shows that the results of this method are similar to those shown in the previous one. The input parameters are listed, and the sample file results are interrupted by the selection list report.

If you do not want the selection list to occur in the middle of the SPS report, the next example shows how you can separate these listings.

### Example Three - Sample Flags with REPORT

Input

```
FILE PAYFILE ...
  NET          5    6    P  2
  EMPLOYEE-NAME 24  20  A
  ...
FILE SAMPFIL ...
*
%SPS1 PAYFILE NET 2000 93848
%SPS2 SAMPFIL PERFORM SELECT-LIST
SELECT-LIST. PROC
  IF SPS-SELECTED EQ 'YES'
    PRINT SELECT-RPT
  END-IF
END-PROC
*
REPORT SELECT-RPT
SEQUENCE EMPLOYEE-NAME
TITLE 01 'SPS SELECTION LIST OF PAYFILE'
LINE EMPLOYEE-NAME NET
```

This example is identical to the one shown in Example Two with one exception. The report contains a SEQUENCE EMPLOYEE-NAME statement. This sequencing forces the output data to be spooled to temporary storage for sequencing. The storage required to spool and sequence the records is obtained automatically.

Due to the sequencing, a line of the report does not immediately print each time SPS-SELECTED contains the value YES. Instead, CA-Easytrieve Plus spools a line of output to temporary storage, and it remains there until all records have been processed. Then CA-Easytrieve Plus sequences the output records by EMPLOYEE-NAME as specified. This allows the SPS routine to finish processing and complete printing before the SELECT-RPT is printed.

Output

```

SPS SAMPLING REPORT
INPUT PARAMETERS

INPUT FILENAME                INFILE
INPUT FIELD                    NET
VALUE OF INPUT FIELD IS      ABS
TARGET VALUE                  20,000.00

SAMPLE FILE

NUMBER OF RECORDS PROCESSED    928
ABS VALUE OF RECORDS PROCESSED 488,886.12
ACT VALUE OF RECORDS PROCESSED 488,886.12
POS VALUE OF RECORDS PROCESSED 488,886.12
NUMBER OF RECORDS IN SAMPLE FILE 24

FILE SAMPFIL WILL BE CREATED
    
```

SPS SELECTION LIST OF PAYFILE PAGE 1

EMPLOYEE-NAME	NET
EMPLOYEE112	952.28
EMPLOYEE146	396.86
EMPLOYEE186	228.74
EMPLOYEE224	504.76
EMPLOYEE263	748.33
EMPLOYEE301	779.90
EMPLOYEE336	953.15
EMPLOYEE37	165.34
EMPLOYEE376	797.40
EMPLOYEE420	701.52
EMPLOYEE456	869.73
EMPLOYEE494	483.01
EMPLOYEE535	752.38
EMPLOYEE572	171.24
EMPLOYEE606	962.78
EMPLOYEE641	626.59
EMPLOYEE682	937.00
EMPLOYEE718	480.92
EMPLOYEE755	388.09
EMPLOYEE76	374.71
EMPLOYEE793	428.20
EMPLOYEE833	295.36
EMPLOYEE872	844.52
EMPLOYEE912	821.08

The output shows the full SPS listing, followed by the employee selection list. The sequencing of the report causes the separation of the listings.

## Stacking Stand-alone Routines

In some applications, you can code multiple stand-alone routines in a single Toolkit execution. This is called stacking of routines. The stacking of routines provides a technique to decrease execution time, reduce the coding of JCL, eliminate the repeated submission of jobs, plus define and create customized complex applications. When you stack stand-alone routines, successive JOB and/or SORT activities are combined after a single library section. This is a form of logic-after-invocation in a stand-alone routine. However, there is a difference between how you can stack and combine inline and stand-alone routines.

The chapter “[Coding Guidelines](#)” explains how you can use more than one inline routine in a single execution and how you can use inline routines with stand-alone routines. These are basic topics that you can uniformly apply to inline and stand-alone routines and are fully discussed in this chapter. However, the stacking of stand-alone routines is a complex issue and is discussed in detail in the next section. Certain limitations exist in stacking stand-alone routines, and these limitations must be fully understood before you use this technique.

Inline routines do not use input files, do not contain CA-Easytrieve Plus REPORT subactivities, and do not produce output files. Because inline routines do not have these processing capabilities, you are able to stack inline routines and combine them with stand-alone routines or any valid CA-Easytrieve Plus logic.

On the other hand, all stand-alone routines use input files, and some use REPORT subactivities and/or produce output files. Some even define their own files for internal use and must, therefore, contain a library section. These processing capabilities make it more difficult to provide the unrestricted flexibility that is available with inline routines.

### Limitations

Use the following guidelines when stacking stand-alone routines:

- Certain stand-alone routines can only be the first of any stacked stand-alone routines. They cannot exist as second or subsequent stand-alone routines.
- When you repeat some stand-alone routines, you cannot change certain parameters in the second and subsequent invocations unless you define them within a different file in the library section.

The first routine limitation has three basic implications:

- First, you cannot use these routines in applications where you repeat the same routine.
- An extension of this limitation is that you cannot invoke two of these routines in a single execution.
- Finally, if you combine any of these routines with other stand-alone routines, you must invoke them as the first stand-alone routine.

The second routine limitation has one basic implication. When using the technique of repeated stacking of the same stand-alone routine, you cannot change certain major fields in the parameter list if the fields are defined in the same file in the library section. However, you can get around this limitation by defining an extra file in the JCL.

## First Routine Limitation

If you stack any of the following routines in a single Toolkit execution, they must be the first routine that the Toolkit execution invokes.

ADDRCMP	MULTDUP
CAVEVAL	SRCECOMP
CBLCNVRT	STOPORGO
DUPTST	STRATIF
ENCRYPT	STRTEVL
GAPCHCK	VERSUS
INTERVL	

## Changing Parameter Limitation

The value of the following parameters must not change when you stack the routine repeatedly, unless you define the parameters in different files in the library section:

ROUTINE	PARAMETERS
CAVSAMP	field
DOLUNIT	field
MULTREG	field1
REGSAM	field1 field2
SIMPREG	field1 field2
SPS	field

To get around this limitation, define a second DDNAME (OS) or DLBL file name (DOS) in the JCL that has the same data set name as the original file. This associates two file names in the JCL with the same physical file. Next, define a file in the library section with the file name of the second file, and use the COPY statement to define the identical fields within that file.

When you need to change one of the preceding parameters in the second invocation statement, specify the second file as the input file and specify the parameter that you want. This uses the same physical file for input in the repeated invocation, but simply refers to it by a different file name. This technique is required only for the parameters listed previously. By defining additional file names, you can use this technique any number of times in a single execution of Toolkit.

If you change any of the preceding fields in a repeated stacking application and you do not use this multiple file technique, unpredictable results occur, and no error message is printed. The report may also contain erroneous information.

The following examples demonstrate the technique of stacking stand-alone routines. The examples also show the limitations that this section explained.

### Example One - Common Stacking Technique

```
FILE PAYFILE ...
  CLIENT          1    3    N
  GROSS-PAY      23    5    P 2
  ...
FILE SAMP104 ...
FILE SAMP109 ...
FILE SAMP211 ...
*
%SPS1 PAYFILE GROSS-PAY 20000 433281
IF CLIENT NE 104
  GO TO JOB
END-IF
%SPS2 SAMP104
*
%SPS1 PAYFILE GROSS-PAY 25000 433281
IF CLIENT NE 109
  GO TO JOB
END-IF
%SPS2 SAMP109
*
%SPS1 PAYFILE GROSS-PAY 18000 433281
IF CLIENT NE 211
  GO TO JOB
END-IF
%SPS2 SAMP211
```

This example demonstrates a common technique for applying stacking. It involves a payroll file with information from different clients. A CLIENT field, which contains a unique number for each client company, identifies each record in the file.

In this example, SPS (Sampling Proportional to Size) routines create a sample of certain clients in the payroll file. The SPS routines are stacked with the screening code section defining the logic to individually select the clients. Notice that the target parameter is different for each execution. This provides the opportunity to customize the sampling based on historical information for each of the clients. Three separate SPS reports and three separate output files containing the appropriate sample records result from this execution.

### Example Two - First Routine Limitation

```
FILE PAYFILE ...
  WEEKLY-GROSS  21  5  P  2
  STATUS-CODE   34  2  N
...
%INTERVL1 PAYFILE GROSS-PAY 100 1000 GRAPH 0 2
%INTERVL2
*
%OCCURS1 PAYFILE GRAPH 0 3
%OCCURS2 STATUS-CODE
```

This example demonstrates a stacking technique with two distribution analysis routines in a single execution. Instead of submitting two separate Toolkit jobs, this example performs both the INTERVL and OCCURS routines in a single execution. The first routine performs an interval analysis of the payroll file, then the routine performs an analysis of the occurrence of the STATUS-CODE field.

Notice that INTERVL is the first of the two routines that this example invokes. This is because INTERVL is a stand-alone routine that has the first routine limitation (see the chart in [First Routine Limitation](#)). If you code OCCURS prior to INTERVL, a syntax error occurs.

### Example Three - Changing Parameter Technique

```
FILE PAYFILE ...
  GROSS-PAY    23  5  P  2
  NET-PAY     28  5  P  2
...
FILE PFILCOP
  COPY PAYFILE
FILE SAMPGRS ...
FILE SAMPNET ...
*
%SPS1 PAYFILE GROSS-PAY 25000 433281
%SPS2 SAMPGRS
*
%SPS1 PFILCOP NET-PAY 20000 433281
%SPS2 SAMPNET
```

In this example, the field parameter is changed in stacked invocations of SPS. The JCL defines the input file twice, once as PAYFILE and once as PFILCOP. These two file names point to the same data set. The library section defines PAYFILE in the same manner as a nonstacked application would normally define PAYFILE. The COPY statement defines the fields in PFILCOP. This provides the same field names for PFILCOP that exist for PAYFILE. The definitions for the two output files follow the COPY statement.

The first SPS routine is invoked with PAYFILE as the input file, GROSS-PAY as the field parameter, and a target of 25,000. The SPS routine is invoked the second time with PFILCOP as the input file, NET-PAY as the field parameter, and a target value of 20,000. Since the value of the field parameter in SPS must not change when the SPS routine is repeatedly stacked, the multiple file technique is required to change the parameter from GROSS-PAY to NET-PAY. This is accomplished by defining PAYFILE and PFILCOP separately in the library section while defining them as the same file in the JCL. Therefore, SPS obtains the data for the field parameter from the same physical file, while specifying in the invocation statements that the data is from separate files.

## Database Use of Toolkit

You can use Toolkit routines to access information in various database structures. The method that you use to access database files depends on whether the routine is an inline or stand-alone routine. If the routine is inline, invoke the Toolkit routine as an integral part of a job activity which is processing the database file. If the routine is stand-alone, Toolkit occasionally requires one additional parameter to inform the routine that it is processing a database file.

In all cases, accessing database files with Toolkit routines requires that you code the CA-Easytrieve Plus statements to properly retrieve the records from the database. It also requires that you code all appropriate checks to assure that RETRIEVE and SELECT statements maintain single complete paths (see the CA-Easytrieve Plus *Reference Guide*). Toolkit routines provide the ability to access information in database files. However, Toolkit does not automate the process of retrieving and constructing records from a database, nor does it check for correct path processing.

## Miscellaneous Techniques

The following miscellaneous advanced coding techniques are discussed next:

- Creating output files in Toolkit routines
- Suppression of Toolkit macro expansions

## Output Files

Many Toolkit routines create an output file (usually in the form of a sample file). In all cases, the library section of the program must contain FILE statements that sufficiently describe the file and the fields necessary to execute the routine.

When coding the attributes for the output file, you must be careful to ensure that the output file has the proper attributes to be able to receive the records written from the input file.

The most common method to ensure that the output file does have the proper attributes is to assign identical attributes to both the input file and the output file. The following example illustrates this method.

### Example One - Fixed Length Files with Identical Lengths

```
FILE INFILE F(100)
*
FILE SAMPFIL F(100)
*
%RANDPCT1 INFILE 10294 2.0 93584
%RANDPCT2 SAMPFIL
```

Both the input and output file are 100-byte fixed length files. In most applications you will want to create a sample file that has identical characteristics as the original file.

However, there are applications where you may want to increase the length of a file or change it from variable to fixed length or vice versa. This example and the next one demonstrate how to get these results.

### Example Two - Fixed Length Files with Different Lengths

```
FILE INFILE F(100)
*
FILE SAMPFIL F(110)
*
%RANDPCT1 INFILE 10294 2.0 93584
%RANDPCT2 SAMPFIL
```

In this example, the records that are selected for sampling are written to an output file that is 10 bytes longer than the input file. The first 100 bytes of each input record are written to the first 100 bytes of the output record. The remaining 10 bytes of the output file have an undetermined value. The output file can have a longer file length than the input file, but it must not have a shorter length.

You could use this method to create space in the sample file to insert additional data such as an audited amount, or a record counter that could be inserted in the screening code section.

### Example Three - Fixed Length Files with Different Lengths

```
FILE INFILE F(100)
*
FILE SAMPFIL F(105)
ORIGINAL-POSITION 101 5 P 0
*
%RANDPCT1 INFILE 10294 2.0 93584
ORIGINAL-POSITION = RECORD-COUNT(INFILE)
%RANDPCT2 SAMPFIL
```

In this example, the additional five bytes of the sample record indicate the position of the record in the original file. The ORIGINAL-POSITION field is defined in the sample file starting at position 101. The screening code section assigns the RECORD-COUNT of the input file to the ORIGINAL-POSITION field. When a given record is selected for sampling, the first 100 bytes are moved to the output buffer, and the final 5 bytes containing the record count do not change. The data in the output buffer is then written to the sample file.

It is important to understand that when you use this technique, Toolkit builds sample records by moving the input data to the output buffer beginning at byte one of the output buffer for the length of the input record. Any data that you insert must be added beyond the length of the input record.

### Example Four - Fixed Length to Variable Length Files

```
FILE INFILE F(100)
*
FILE SAMPFIL V(104)
*
%RANDPCT1 INFILE 10294 2.0 93584
%RANDPCT2 SAMPFIL
```

This example demonstrates the creation of a variable length sample file from a 100-byte fixed-length input file. The sample file is defined as 104 bytes to accommodate the four bytes for the record descriptor word. When creating a variable-length sample file from a fixed-length input file, make sure that the length of the sample file is at least four bytes greater than the input file.

### Example Five - Variable Length to Fixed Length Files

```
FILE INFILE V(104)
*
FILE SAMPFIL F(100)
*
%RANDPCT1 INFILE 10294 2.0 93584
%RANDPCT2 SAMPFIL
```

This example demonstrates the creation of a 100-byte fixed-length sample file from a 104-byte variable-length input file. The sample file is defined as 100 bytes because the 104-byte length of the input file includes four bytes for the record descriptor word, which a fixed-length file does not use. When creating a fixed-length sample file from a variable-length input file, make the minimum length of the sample file four bytes less than the input file.

## Output File Summary

When the input file is a fixed-length file, the output file may be:

- Fixed-length with a record length greater than or equal to the record length of the input file
- Variable-length with a record length greater than or equal to the record length of the input file plus four

When the input file is a variable-length file, the output file may be:

- Fixed-length with a record length greater than or equal to the length of the input file minus four
- Variable-length with a record length greater than or equal to the record length of the input file

## Suppression of Macro Expansions

To suppress the expansion of the Toolkit statements that comprise the various macros, specify the CA-Easytrieve Plus LIST NOMACROS statement. For example:

```
FILE PAYFILE
  EMPLOYEE-NAME      1   20  A
  NET-PAY             21   5  P  2
  SSN                 26   9  N
  HIRE-DATE          35   6  A
*
JOB INPUT PAYFILE
%NUMTEST NET-PAY 'NET-PAY NOT NUMERIC' EMPLOYEE-NAME
*
LIST NOMACROS
*
%DATEVAL HIRE-DATE MMDDYY
IF DATEVAL-FLAG EQ 'NO'
  DISPLAY 'INVALID HIRE DATE FOR ' EMPLOYEE-NAME  +
        '   DATE= ' HIRE-DATE
END-IF
*
LIST MACROS
*
%NUMTEST SSN 'SSN NOT NUMERIC' EMPLOYEE-NAME
```

This example demonstrates the use of the LIST NOMACROS statement to suppress the listing of the Toolkit routines. The first NUMTEST routine expands all macro statements because the default value for the Toolkit system is LIST MACROS. Prior to the DATEVAL routine, the LIST NOMACROS statement suppresses the listing of the DATEVAL macro statements. The only statements that are in the listing for DATEVAL are the five statements listed after the LIST NOMACROS statement. Then, the LIST MACROS statement turns the listing back on prior to the second NUMTEST routine.

You can use the LIST NOMACROS and LIST MACROS statements at any time to turn the macro expansion listings on and off. You can even use these statements in the screening code section to turn the remainder of a listing on or off.

## Functional Chart of Inline and Stand-alone Routines

The functional chart of inline and stand-alone routines lists the general and statistical routines. The chart summarizes appropriate functional processing information for each routine, and it indicates whether a routine:

- Is inline, stand-alone DISPLAY, or stand-alone REPORT
- Can be used in a database application
- Contains a FILE statement
- Uses the sample flag technique
- Contains stacking limitations

In the chart, IL indicates an inline routine, SA-D a stand-alone DISPLAY routine, and SA-R a stand-alone REPORT routine. YES and NO indicate that a certain feature or limitation applies, and N/A indicates that it is not applicable. The classifications other than Type and Database apply only to stand-alone routines, so all inline routines are listed as N/A. The classifications are summarized after the chart and are discussed in detail earlier in this chapter.

The first column of the chart indicates the type of the routine, which controls the manner in which you can combine CA-Easytrieve Plus LOGIC with Toolkit routines. In the case of stand-alone routines, it specifically controls the coding in the logic-after-invocation section.

The DB column indicates whether you can use a routine in a database application.

The FILE column indicates whether a stand-alone routine defines its own files. This controls the coding of the logic-before-invocation section.

The Sample Flag column indicates whether the stand-alone routine provides a parameter for the sampling flag technique. This provides the ability to identify records that have been selected for sampling.

The stacking limitations column indicates whether a stand-alone routine is limited in its ability to be combined with itself or other stand-alone routines in a single execution of Toolkit.

**Functional Chart**

<b>Routine</b>	<b>Type</b>	<b>DB</b>	<b>FILE</b>	<b>Flag</b>	<b>Limitation</b>
ADDRCMP	SA-R	YES	YES	NO	YES
ALPHACON	IL	YES	N/A	N/A	N/A
ALPHAGEN	IL	NO	N/A	N/A	N/A
APR	IL	YES	N/A	N/A	N/A
ATTPCT	IL	YES	N/A	N/A	N/A
ATTSAMP	SA-D	YES	NO	YES	NO
BADGEN	IL	NO	N/A	N/A	N/A
CAVEVAL	SA-D	NO	YES	NO	YES
CAVSAMP	SA-D	YES	NO	YES	YES
CBLCNVRT	SA-R	NO	YES	NO	YES
CONVAE	IL	YES	N/A	N/A	N/A
CONVEA	IL	YES	N/A	N/A	N/A
DATECALC	IL	YES	N/A	N/A	N/A
DATECONV	IL	YES	N/A	N/A	N/A
DATEGEN	IL	NO	N/A	N/A	N/A
DATEVAL	IL	YES	N/A	N/A	N/A
DAYSAGO	IL	YES	N/A	N/A	N/A
DAYSAGOL	IL	YES	N/A	N/A	N/A
DAYSALC	IL	YES	N/A	N/A	N/A
DECRYPT	IL	YES	N/A	N/A	N/A
DISCPCT	IL	YES	N/A	N/A	N/A
DISCSMP	SA-D	YES	NO	YES	NO
DIVIDE	IL	YES	N/A	N/A	N/A
DOLUNIT	SA-D	YES	NO	YES	YES
DUPTEST	SA-R	YES	YES	NO	YES
EACHNTH	IL	YES	N/A	YES	N/A
ENCRYPT	SA-D	YES	YES	NO	YES
EXPO	IL	YES	N/A	N/A	N/A
FILECOMP	SA-D	NO	NO	NO	NO

<b>Routine</b>	<b>Type</b>	<b>DB</b>	<b>FILE</b>	<b>Flag</b>	<b>Limitation</b>
FILEGEN	IL	NO	N/A	N/A	N/A
FLDVALR	SA-R	YES	NO	NO	NO
FLDVALT	SA-R	YES	NO	NO	NO
FLDVALV	SA-R	YES	NO	NO	NO
GAPCHCK	SA-R	YES	YES	NO	YES
GETDATE	IL	YES	N/A	N/A	N/A
GETDATEL	IL	YES	N/A	N/A	N/A
INTERVL	SA-R	YES	YES	NO	YES
INTSAMP	SA-D	YES	NO	YES	NO
MULTDUP	SA-R	YES	YES	NO	YES
MULTREG	SA-D	YES	NO	NO	YES
NUMGEN	IL	NO	N/A	N/A	N/A
NUMTEST	IL	YES	N/A	N/A	N/A
OCCURS	SA-R	YES	NO	NO	NO
POPCOUNT	IL	YES	N/A	N/A	N/A
POPSIZE	SA-D	YES	NO	NO	NO
RANDOM	IL	YES	N/A	N/A	N/A
RANDPCT	SA-D	YES	NO	YES	NO
RANDSPAN	IL	YES	N/A	N/A	N/A
RANDXCT	SA-D	YES	NO	YES	NO
REGEVAL	SA-D	NO	NO	NO	NO
REGSAM	SA-D	NO	NO	NO	YES
REGSAMP	SA-D	YES	NO	YES	NO
SIMPREG	SA-D	YES	NO	NO	YES
SPS	SA-D	YES	NO	YES	YES
SQRT	IL	YES	N/A	N/A	N/A
SRCECOMP	SA-R	NO	YES	NO	YES
STDDEV	IL	YES	N/A	N/A	N/A
STOPORGO	SA-D	NO	YES	YES	YES
STRATIF	SA-R	YES	YES	NO	YES

<b>Routine</b>	<b>Type</b>	<b>DB</b>	<b>FILE</b>	<b>Flag</b>	<b>Limitation</b>
STRTEVL	SA-D	NO	YES	NO	YES
TIMECONV	IL	YES	N/A	N/A	N/A
UNBYTE	IL	YES	N/A	N/A	N/A
VARPCT	IL	YES	N/A	N/A	N/A
VARSAVP	SA-D	YES	NO	YES	NO
VERSUS	SA-R	YES	YES	NO	YES
WEEKDAY	IL	YES	N/A	N/A	N/A

## Summary of Techniques

Inline Routines (IL) can be combined with other Toolkit routines or CA-Easytrieve Plus logic and are unrestricted, or unlimited in usage.

### Stand-alone DISPLAY Routines (SA-D)

- Use DISPLAY statements to create listings.
- Can accommodate user-coded procedures in the logic-after-invocation section.
- Use special-name REPORT procedures only if you code a REPORT subactivity.

### Stand-alone REPORT Routines (SA-R)

- Use REPORT subactivities to create listings.
- Cannot accommodate procedures in the logic-after-invocation section.
- Can use special-name REPORT procedures if the routine does not already use them (see [Special-Name REPORT Procedures](#) earlier in this chapter).

### Database Routines (DB)

- Inline routines can be invoked as an integral part of any job activity that is processing a database file. They require no special processing considerations.
- Some stand-alone routines require an additional parameter to inform the routine that it is processing a database file.

- Some stand-alone routines can access database and nondatabase files without the specification of additional parameters.
- Because of the nature of the processing of some stand-alone routines, or because of the operational characteristics that they use, database use of some stand-alone routines is not possible.

### **FILE Limitations (FILE)**

- Routines that define their own files cannot use the logic-before-invocation section.

### **Sample Flags**

- Most routines that create sample files provide the capability to test a reserved field to determine if a record was selected for the sample file.
- The PERFORM procname parameter identifies routines with sample flags.

### **Stacking Limitations**

- Some stand-alone routines can only be the first of any stacked stand-alone routines.
- Some stand-alone routines have limitations regarding the specification of parameters when the routine is repeatedly stacked in a single execution of Toolkit.

### **Logic-Before-Invocation Section of Stand-alone Routines**

- Located after the library section and before the first Toolkit macro invocation statement.
- Used to preprocess input data, merge or combine input files, or for other pre-invocation purposes.
- Routines that define their own files cannot use the logic-before-invocation section. The FILE column in the functional chart of Inline Stand-alone Routines (earlier in this section) identifies these routines.

### **Logic-After-Invocation Section of Stand-alone Routines**

- Located after the final Toolkit macro invocation statement.
- Use to perform customized processing with the routine and/or to customize the listing.
- Procedures are performed and/or REPORTs are printed in the screening code section.

- Procedures and/or REPORTs are coded in the logic-after-invocation section of stand-alone routines.
- If procedures and REPORTs are used, the procedures must precede the definition of any REPORTs.
- Any special-name REPORT procedures must be coded following the associated REPORT.

# Keywords

Two types of keywords are used in the execution of CA-Easytrieve Plus Toolkit:

- CA-Easytrieve Plus
- Toolkit

The CA-Easytrieve Plus keywords are those keywords that are associated with the CA-Easytrieve product and are found in the appendix “Keywords” in the CA-Easytrieve Plus *Reference Guide*.

The Toolkit keywords are reserved words used by the Toolkit routines.

## Toolkit Keywords

Toolkit keywords cannot be used for the following:

- Field names
- File names
- Report names
- Procedure names
- Statement labels

The following table shows the list of Toolkit keywords and the routines in which they are used:

<b>Keyword</b>	<b>Routine</b>
KEYSB	FILECOMP
NOFILE	Most routines with an output file
PRIKEYS	FILECOMP
SECKEYS	FILECOMP
SUMMARY	DUPTEST
THRESHOLD	Date routines



# Index

## %

---

- % inline routine prefix, 3-3
- % JIFREC macro invocation prefix, 7-3, 7-27, 7-28
- % stand-alone routine prefix, 3-6, 3-7

## A

---

- A routine, SMF audit routines, 6-3
- abend
  - by abend code routine (JIF), 7-50
  - by program routine (JIF), 7-52
  - by programmer routine (JIF), 7-54
- ADDRCMP routine, 2-9, 4-1
- ALPHACON routine, 2-9, 4-5
- ALPHAGEN routine, 2-7, 2-8
  - description, 4-8
- application trend analysis routine (JIF), 7-48
- APR routine, 2-9, 4-10
- audit file statistical report routine (JIF), 7-56

## B

---

- B routine, SMF audit routines, 6-3
- BADGEN routine, 2-7, 2-8
  - description, 4-13
- bar graph
  - deviation keywords, 8-25
  - deviation, function, 8-25
  - function, 8-6
  - keywords, 8-7

- binary field processing
  - example, 5-50
  - UNBYTE routine, 5-49

## C

---

- CA-Easytrieve Plus
  - coding, 3-1
  - procedures, 9-2
  - read input exit (JIF), 7-2
  - statements, use in Toolkit, 3-1
- CA-Easytrieve, language
  - use in Toolkit, 1-1
- calculation
  - effective annual percentage rate. *See* APR routine testing, 3-2
- capabilities, 1-2
- changing parameter
  - limitation, stacking stand-alone routines, 9-18
  - technique, stacking stand-alone routines, 9-20
- chart, functional chart of inline and stand-alone routines, 9-25
- class structure analysis routine (JIF), 7-44
- combined costs, EXCOSTS, 7-67
- common stacking technique, 9-19
- compliance testing, EACHNTH, 5-1
- consolidated file. *See* JIF
- CONTROL
  - SMF005, 6-11
- control breaks, 8-6
- CONTROL statement
  - keywords, A-1
- CONVAE routine, 2-9, 4-16
- CONVEA routine, 2-9, 4-18

---

conventions, notational, 1-3

#### CPU

- charging, EXCHGCPU, 7-62
- time distribution routine (JIF), 7-45
- weighting, EXWGTCPU, 7-61

#### creating

- an input file from two or more files, stand-alone routines, 9-11
- output files in routines, 9-21

CREDTAB, credit/budget table, 7-68, 7-69

## D

---

#### data

- processing invoice, JIFBEX06, 7-76
- validation
  - FLDVALR, 5-12
  - FLDVALT, 5-15

#### data center

- cost recovery, JIFBEX04, 7-73

#### data validation

- FLDVALV, 5-19
- GAPCHCK, 5-22

#### database

- routines, summary of techniques, 9-29

#### date and time routines, 2-2

DATECALC routine, 2-2  
description, 4-20

DATECONV routine, 2-3, A-1  
description, 4-23

DATEGEN routine, 2-7, 2-8  
description, 4-26

DATEVAL routine, 2-2  
description, 4-29

DAYSAGO routine, 2-3  
description, 4-32

DAYSAGOL routine  
description, 4-35

DAYSCALC routine, 2-3, 4-38

DEFINE statement  
keywords, A-1

DEPTTAB, department table, 7-48, 7-68

detail charge audit, JIFBEX02, 7-70

deviation bar graph, 8-25. *See also* bar graph

#### DISPLAY statement

- CA-Easytrieve Plus, 3-4
- keywords, A-1

DIVIDE routine, 2-3, 4-41

DLI, 1-1

DOLUNIT routine  
description, 4-42

DUPTTEST routine, 2-2, A-1  
description, 4-49

## E

---

EACHNTH routine, 2-9, 5-1

effective annual percentage rate, 4-10

END-IF statement, 3-9

#### environment

- EXITSTR, 8-2

error messages, syntax, 3-9

#### example billing routines

- created from the SMF data, 7-68
- data center cost recovery, 7-73
- data processing invoice, 7-76
- detail charge audit, 7-70
- invoice ledger, 7-69
- monthly utilization by cost center, 7-75
- TSO user charging report, 7-77

EXCHGCPU, CPU charging routine, 7-62

EXCHGIO - I/O charging routine, 7-63

EXCHGTPG, TPUT and TGET charging routine, 7-65

EXCHGUR, unit record device charging routine, 7-66

EXCOSTS, combined costs routine, 7-67

exit facility, JIF, 7-2

EXITSTR, 8-2

EXPO routine, 2-3, 5-3

EXWGTCPU, CPU weighting routine, 7-61

EXWGTPTY, job priority weighting routine, 7-60

---

## F

### fields

- comparing, 2-4
- conversion to numeric. *See* ALPHACON routine
- definition, 1-1
- FILE-COUNT (FILEGEN), 4-15
- USER-RANDOM (FILEGEN), 4-15, 4-16

### FILE

- limitations, summary of techniques, 9-29
- statement
  - graphing facility
    - input file, 8-3, 8-4
    - keyword file, 8-2, 8-3

file compare facility, 2-4

### FILE statement

- keywords, A-1

### FILECOMP routine

- description, 2-4, 5-6

### FILEGEN routine, 2-7, 2-8

- description, 5-11
- FILE-COUNT field, 4-15
- USER-RANDOM field, 4-15, 4-16

### files

- comparing fields, 2-4, 2-5
- input
  - graphing facility, 8-1
  - screening data, 3-7

### files, input

- layout, 1-1

### files, output

- types produced, 1-1

first routine limitation, stacking stand-alone routines, 9-18, 9-20

### FLDVALR routine, 5-12

### FLDVALT routine, 5-15

### FLDVALV routine, 5-19

format stand-alone routine, 3-5

functional chart of inline and stand-alone routines, 9-25

---

## G

GAPCHCK routine, 2-2, 5-22

generalized routines, 2-1

GETDATE routine, 2-3, 5-24

GETDATEL routine, 5-25

GO TO JOB statement, 3-9

### graphing

#### facility

- coding required, 8-2
  - histogram, 8-15
  - invocation statement, 8-3
  - keyword file, 8-3, 8-4, 8-5
  - OS JCL example, 8-29
  - plot graph, 8-20
  - requirements, 8-3
  - standard bar graph, 8-6
  - types of graphs, 8-1
- frequency, 3-9

---

## H

### hexadecimal listing

- test data, 2-7

histogram, 8-15

host language, Toolkit. *See* CA-Easytrieve Plus

hourly throughput analysis routine (JIF), 7-38

hourly turnaround routine (JIF), 7-41

---

## I

I/O charging, EXCHGIO, 7-63

IDMS, 1-1

IF statement, 3-9

IMS, 1-1

IMS/DLI, 1-1

### inline routines

- characteristics, 3-1
- functional chart, 9-25
- invoking, 3-3
- requirements, 3-2
- stacking of, 9-17
- summary of techniques, 9-28
- use of procedures with, 9-2, 9-3

integrity tests, 2-1

---

invoice ledger, JIFBEX01, 7-69

invoking SMF audit routines, 6-2

IPL reporting, SMF000, 6-4

## J

---

JIF, 7-1

audit file, 7-9, 7-20, 7-56

audit record, 7-21

audit report, 7-10, 7-23

consolidated record, 7-9, 7-25, 7-27

duplicate records, 7-25

function of, 7-1

read input exit, 7-2, 7-25

rerun records, 7-25

SMF record selection, 7-5

statistical routines, 7-28

user exit facility

EXIT1, 7-5, 7-24, 7-25

EXIT2, 7-5, 7-24, 7-25

function of, 7-2

JIFBEX01, invoice ledger routine, 7-69

JIFBEX02, detail charge audit routine, 7-70

JIFBEX03 - job charge summary routine, 7-72

JIFBEX04, data center cost summary routine, 7-73

JIFBEX05, monthly utilization by cost center routine, 7-75

JIFBEX06, data processing invoice routine, 7-76

JIFBEX07, TSO user charging report routine, 7-77

JIFOPTS, options module, 7-2, 7-4

JIFRDREX, 7-2

JIFSEL, 7-2, 7-5

job charge summary - JIFBEX03, 7-72

Job Information Facility (JIF). *See* JIF

JOB or SORT activities in logic-after-invocation of stand-alone routines, 9-5

job priority weighting, EXWGTPTY, 7-60

JOB statement

graphing facility, 8-2

stand-alone routine, 3-7

## L

---

library

graphing facility, 8-2

section, stand-alone routine, 3-6

logic-after-invocation

description, 3-8

of stand-alone routines

creating an input file, 9-11

summary of techniques, 9-30

use of procedures with, 9-2

use with

JOB or SORT activities, 9-5

special-name REPORT routines, 9-5

stand-alone routines, 9-4

logic-before-invocation

of stand-alone routines

description, 3-6, 9-9

pre-calculation, 9-10

summary of techniques, 9-29

use of procedures with, 9-2

## M

---

macro

expansions, suppression of, 9-21, 9-24

facility

graphing, 8-1

inline routine, 3-3

invoking a macro, 3-3

stand-alone routine, 3-6

SMF record field definitions, 6-39

mapping convention, SMF audit routines, 6-2

miscellaneous routines, 2-8

monthly utilization by cost center, JIFBEX05, 7-75

MULTDUP routine, 2-2, 5-26

## N

---

numeric data

random number generator. *See* RANDOM, RANDSPAN

numeric routines, 2-3

NUMGEN routine, 2-7, 2-8

description, 5-30

---

NUMTEST routine, 2-2, 5-32, 9-3

## O

---

object modules

comparing, 2-4

OS Job Information Facility (JIF). *See* JIF

OSJIF01 routine, 7-29

OSJIF02 routine, 7-31

OSJIF03 routine, 7-33

OSJIF04 routine, 7-34

OSJIF05 routine, 7-35

OSJIF06 routine, 7-37

OSJIF07 routine, 7-38

OSJIF08 routine, 7-39

OSJIF09 routine, 7-41

OSJIF10 routine, 7-42

OSJIF11 routine, 7-44

OSJIF12 routine, 7-45

OSJIF13 routine, 7-46

OSJIF14 routine, 7-48

OSJIF15 routine, 7-50

OSJIF16 routine, 7-52

OSJIF17 routine, 7-54

OSJIF18 routine, 7-56

OSJIF19 routine, 7-57

output files, creation of in routines, 9-21

## P

---

page peaking periods routine (JIF), 7-42

parameters

SMF000 IPLs, 6-4

SMF005, 6-11

SMF006, control forms output, 6-16

SMF007, SMF data lost, 6-18

SMF014, data set activity, 6-19

SMF017, scratched data sets, 6-23

SMF018, renamed data sets, 6-26

SMF020, job initiations, 6-28

SMF049, JES2 and JES3 integrity, 6-30

SMF062, VSAM data sets opened, 6-31

SMF067, deleted VSAM entries, 6-33

SMF068 renamed VSAM data sets, 6-36

PARM statement

keywords, A-1

performance

group

/priority/class routine (JIF), 7-37

profile routine (JIF), 7-34

summary routine (JIF), 7-35

objective summary routine (JIF), 7-31

placement

of procedures in inline routines, 9-4

of procedures in stand-alone routines, 9-6

of reports in stand-alone routines, 9-6

plot graph

function, 8-20

keywords, 8-21

precalculation techniques, logic-before-invocation of stand-alone routines, 9-10

PRIKEYS, 2-4

parameters, 5-7

procedures

placement of in inline routines, 9-4

placement of in stand-alone routines, 9-6

with inline routines, 9-2, 9-3

with logic-after-invocation, 9-2

with logic-before-invocation, 9-2

with sample flag techniques, 9-12

with sampling flag technique, 9-2

with stand-alone routines, 9-4

publications

CA-Easytrieve, 1-2

mainframe, 1-2

## R

---

random number generator. *See* RANDOM, RANDSPAN

RANDOM routine, 2-3, 5-34

RANDSPAN routine, 2-3, 5-35

records

comparing, 2-4

data lost, 7-6, 7-20, 7-23

- duplicate, SMF (JIF facility), 7-25
- IPL, 7-6, 7-9, 7-23
- JIF, 7-1
  - abend, 7-50
  - audit file, 7-3, 7-9
  - consolidated
    - audit report, 7-23
    - EXIT2, 7-25
    - file, 7-2, 7-3, 7-9
    - record, 7-10
    - SMF, 7-5, 7-7
  - CPU time, 7-42
  - data lost, 7-23
  - duplicate, 7-25
  - dynamic DD, 7-6
  - I/O count, 7-34
  - IPL, 7-6, 7-23
  - JES2, 7-6
  - JES3, 7-6
  - job class, 7-37
  - job priority, 7-37
  - job purge, 7-7
  - lost data, 7-6, 7-20
  - orphan, 7-23
  - performance group, 7-35
  - rerun, 7-23, 7-25
  - step information, 7-52
  - TSO, 7-2, 7-6
- keys, 2-4
- REPORT
  - SMF005, 6-11
  - SMF006, 6-16
  - SMF007, 6-18
  - SMF014, 6-19
  - SMF017, 6-23
  - SMF018, 6-26
  - SMF020, 6-28
  - SMF049, 6-30
  - SMF062, 6-31
  - SMF067, 6-33
  - SMF068, 6-36
  - statement
    - CA-Easytrieve Plus, 3-4
- REPORT statement
  - keywords, A-1
- reports
  - inline routines, 3-3
  - placement of in stand-alone routines, 9-6

- routines
  - stand-alone, format, 3-5
  - stand-alone-DISPLAY, 3-4
  - stand-alone-REPORT, 3-4
  - that use sample flags, 9-12

---

## S

---

- sample
  - flag techniques
    - routines that use, 9-12
    - summary of, 9-29
    - use of procedures with, 9-2, 9-12
    - with DISPLAY, 9-13
    - with REPORT, 9-14, 9-16
  - routine, EACHNTH, 5-1
- screening
  - facility, use of, 3-7
  - logic, graphing facility, 8-2, 8-6
  - processing logic, 4-46
- SECKEYS, 2-4
  - parameters, 5-7
- service requirements routine (JIF), 7-39
- SMF
  - audit routines, 6-2, 6-3
  - record field definitions, 6-39
  - record type field definition routines, 6-3
- SMF000, IPLs, 6-4
- SMF004
  - abnormal step terminations, 6-6
  - control options, 6-9
- SMF005, abnormal job terminations, 6-11
- SMF006, control forms output, 6-16
- SMF007, SMF data lost, 6-18
- SMF014, data set activity, 6-19
- SMF017, scratched data sets, 6-23
- SMF018, renamed data sets, 6-26
- SMF020, job initiations, 6-28
- SMF049, JES2 and JES3 integrity, 6-30
- SMF062, VSAM data sets opened, 6-31
- SMF067, deleted VSAM entries, 6-33
- SMF068, renamed VSAM data sets, 6-36

---

SMFCNT, frequency distribution of SMF records, 6-37

## SORT

activities in logic-after-invocation section of stand-alone routines, 9-5

keywords, A-1

SMF005, 6-11

SMF014, 6-19

SMF017, 6-23

SMF018, 6-26

## sorting

bar graph data, 8-6

graphing facility data, 8-4

histogram graph data, 8-15

JIF facility, 7-5

## source modules

comparing, 2-4

## special-name REPORT procedure

use with a stand-alone-REPORT routine, 9-8

use with logic-after-invocation, 9-5

use with stand-alone routines, 9-5

## SQL, 1-1

SQRT routine, 2-3, 5-37

SRCECOMP routine, 2-4, 5-38

## stacking

inline routines, 9-17

limitations, summary of techniques, 9-29

SMF audit routines, 6-2

stand-alone routines

changing parameter limitation, 9-18

changing parameter technique, 9-20

common stacking technique, 9-19

description, 9-17

first routine limitation, 9-18, 9-20

limitations, 9-17

## stand-alone routines

(display and report)

difference, 9-4

general information, 3-4

characteristics, 3-1, 3-4

format, 3-5

functional chart, 9-25

invoking, 3-6, 3-7

JOB or SORT activities, 9-5

logic-after-invocation, creating an input file, 9-11

logic-before-invocation, 9-9

placement of procedures and reports, 9-6

pre-calculation techniques, 9-10

special-name REPORT procedure, 9-8

stacking of. *See* stacking stand-alone routines

use of with procedures, 9-4

use with special-name REPORT procedures, 9-5

stand-alone-DISPLAY routines, summary of techniques, 9-28

stand-alone-REPORT routines, summary of techniques, 9-28

## standard

bar graph. *See* bar graph

STDDEV routine, 2-3, 5-41

SUMMARY keyword, A-1

suppression of macro expansions, 9-21

System Management Facilities (SMF). *See* JIF

---

## T

### tables and arrays

credit/budget table (JIF), 7-68, 7-69

department table (JIF), 7-48, 7-68

tape allocation routine (JIF), 7-46

### test data

alphanumeric, 2-8

calculated, 4-14

dates, 2-8

FILE-COUNT field, 4-15

generation facility, 2-7

generation routines, 2-8

invalid, 2-8, 4-15

numeric, 2-8, 4-14

USER-RANDOM field, 4-15, 4-16

test data generation, 2-7

routines, 2-8

TIMECONV routine, 2-3, 5-47

### Toolkit

host language. *See* CA-Easytrieve Plus

keywords, A-1

routines

(first invocation), stand-alone routine, 3-6

coding. *See* stand-alone and inline

types. *See* general and statistical

TPUT and TGET charging, EXCHGTPG, 7-65

---

## T

- reporting on, 7-5, 7-6
- session analysis routine (JIF), 7-57
- TGETS, 7-59
- TPUTS, 7-59
- user charging report, JIFBEX07, 7-77

## U

---

- UNBYTE routine, 2-9, 5-49
- unit record device charging, EXCHGUR, 7-66
- user code
  - inline routines, 3-3
- USER-RANDOM field, 4-16

## V

---

- VERSUS routine
  - example, 3-9

## W

---

- WEEKDAY routine, 2-3, 5-51
- weighting and costing examples
  - combined costs, 7-67
  - CPU charging, 7-62
  - example billing routines, 7-68
  - I/O charging, 7-63
  - TPUT and TGET charging, 7-65
  - unit record device charging routine, 7-66
- working storage, inline routines, 3-3
- workload trend analysis routine (JIF), 7-33

## X

---

- XFLAGS. *See* JIF